MIT/LCS/TR-2

SIR: A COMPUTER PROGRAM FOR SEMANTIC INFORMATION RETRIEVAL

Betram Raphael

June 1964

SIR: A COMPUTER PROGRAM FOR SEMANTIC INFORMATION RETRIEVAL

by

BERTRAM RAPHAEL

B.S., Rensselaer Polytechnic Institute
(1957)

M.S., Brown University
(1959)

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

at the

MASSACHUSETTS INSTITUTE OF
TECHNOLOGY

June, 1964

Signature of Author ....
Department of Mathematics, April 8, 1964

Certified by ....
Thesis Supervisor

....
Chairman, Departmental Committee
on Graduate Students

Acknowledgement

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). Associated preliminary research was supported in part by the RAND Corporation, Santa Monica, California, and in part by Bolt, Beranek, and Newman, Inc., Cambridge, Massachusetts. The author expresses his gratitude to Prof. Marvin Minsky, for his supervision of this thesis; to Dr. Victor Yngve and Prof. Hartley Rogers, Jr., for their criticisms and suggestions; and to his wife Anne for her unfailing confidence and encouragement, without which this thesis could not have been completed.

SIR: A COMPUTER PROGRAM FOR
SEMANTIC INFORMATION RETRIEVAL

by
BERTRAM RAPHAEL

Submitted to the Department of Mathematics on April 8, 1964, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

ABSTRACT

SIR is a computer system, programmed in the LISP language, which accepts information and answers questions expressed in a restricted form of English. This system demonstrates what can reasonably be called an ability to "understand" semantic information. SIR's semantic and deductive ability is based on the construction of an internal model, which uses word associations and property lists, for the relational information normally conveyed in conversational statements.

A format-matching procedure extracts semantic content from English sentences. If an input sentence is declarative, the system adds appropriate information to the model. If an input sentence is a question, the system searches the model until it either finds the answer or determines why it cannot find the answer. In all cases SIR reports its conclusions. The system has some capacity to recognize exceptions to general rules, resolve certain semantic ambiguities, and modify its model structure in order to save computer memory space.

Judging from its conversational ability, SIR is more "intelligent" than any other existing question-answering system. The author describes how this ability was developed and how the basic features of SIR compare with those of other systems.

The working system, SIR, is a first step toward intelligent man-machine communication. The author proposes a next step by describing how to construct a more general system which is less complex and yet more powerful than SIR. This proposed system contains a generalized version of the SIR model, a formal logical system called SIR1, and a computer program for testing the truth of SIR1 statements with respect to the generalized model by using partial proof procedures in the predicate calculus. The thesis also describes the formal properties of SIR1 and how they relate to the logical structure of SIR.

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

TABLE OF CONTENTS

Chapter I: Introduction

A. The Problem.

This report discusses the problem of developing a computer* which "understands." The method of study involved the construction of a computer system having certain cognitive abilities and exhibiting some human-like conversational behavior.

This computer system is called "SIR" -- Semantic Information Retriever. The conversation presented in Fig. 1 between a person (identified by "***.") and SIR illustrates some of the system's capabilities. Although it is capable of "understanding" statements dealing with set relations, part-whole relations, ownership, and certain spatial relations, SIR has no initial built-in associations for nouns like "John," "boy," and "table."

"Understanding" is difficult to define precisely. The present study makes the following specific assumptions and observations:

i) "Understanding" can be demonstrated by dialogue, which requires the participants to indicate an awareness of the meaning of their subject matter. Therefore a computer should be considered able to "understand" if it can converse intelligently, i.e., if it can remember what it is told, answer questions and make responses which a human observer considers reasonable.

* The term "computer" refers to the combination of digital computer hardware and operating computer program.

```
(***.     EVERY BOY IS A PERSON)
(I UNDERSTAND)

(***.     A FINGER IS PART OF A HAND)
(I UNDERSTAND)

(***.     EACH PERSON HAS TWO HANDS)
(THE ABOVE SENTENCE IS AMBIGUOUS ** PLEASE RE-PHRASE IT)

(***.     THERE ARE TWO HANDS ON EACH PERSON)
(I UNDERSTAND)

(***.     HOW MANY FINGERS DOES JOHN HAVE Q)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(I DON*T KNOW WHETHER FINGER IS PART OF JOHN)

(***.     JOHN IS A BOY)
(I UNDERSTAND)

(***.     HOW MANY FINGERS DOES JOHN HAVE Q)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
((HOW MANY FINGER PER HAND Q))

(***.     EVERY HAND HAS 5 FINGERS)

(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(I UNDERSTAND)

(***.     HOW MANY FINGERS DOES JOHN HAVE Q)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(THE ANSWER IS 10)

(***.     HOW MANY AUTOMOBILES DOES JOHN HAVE Q)
(THE ABOVE SENTENCE IS AMBIGUOUS ** PLEASE RE-PHRASE IT)

(***.     WHO IS PRESIDENT OF THE UNITED STATES Q)
(STATEMENT FORM NOT RECOGNIZED)

(***.     THE BOY IS JUST TO THE LEFT OF THE TABLE)
(G02840 IS A TABLE)
(I UNDERSTAND)

(***.     THE LAMP IS JUST TO THE LEFT OF THE TABLE)
(G02841 IS A LAMP)
(THE ABOVE STATEMENT IS IMPOSSIBLE)

(***.     THE TABLE IS TO THE RIGHT OF THE CHAIR)
(G02842 IS A CHAIR)
(I UNDERSTAND)

(***.     WHAT IS THE RELATIVE POSITION OF A PERSON Q)
(THE LEFT-TO-RIGHT ORDER IS AS FOLLOWS)
(CHAIR (BOY TABLE))
```

FIGURE 1: SAMPLE CONVERSATION

<u>Note:</u> I am concerned here with the computer's internal information representation and retrieval techniques. For this purpose I assume that abstract "words" are the basic signal unit. There is no need to be concerned with speech recognition, sensory receptors, or other problems involving the physical nature of the communication channel and signals.

ii) In addition to echoing, upon request, the facts it has been given, a machine which "understands" must be able to recognize the logical implications of those facts. It also must be able to identify (from a large data store) facts which are relevant to a particular question.

iii) The most important prerequisite for the ability to "understand" is a suitable internal representation, or model, for stored information. This model should be structured so that information relevant for question-answering is easily accessible. Direct storage of English text is not suitable since the structure of an English statement generally is not a good representation of the meaning of the statement. On the other hand, models which are direct representations of certain kinds of relational information usually are unsuited for use with other relations. A general-purpose "understanding" machine should utilize a model which can represent semantic content for a wide variety of subject areas.

SIR is a prototype of an "understanding" machine. It demonstrates how these conversational and deductive abilities can be obtained through use of a suitable model. Later chapters will describe the model and the SIR program, how they were developed, how they are used, and how they can be extended for future applications.

B. Where the Problem Arises.

The need for computers which "understand" arises in several areas of computer research. Some examples follow.

1) Information retrieval: The high speeds and huge memory capacities of present computers could be of great aid in scanning scientific literature. Unfortunately, high-speed search is useless unless the searcher is capable of recognizing what is being searched for; and existing computer systems for information retrieval use too crude techniques for specifying and identifying the objects of the search.

Information retrieval systems generally provide either document retrieval or fact retrieval. Document retrieval programs usually depend upon a human pre-assignment of "descriptors" to the documents. A user of the system may know the list of descriptors but cannot know precisely what the descriptors meant to the cataloguer. It is difficult for the user to determine what the semantic interactions between the descriptors are and how these interactions help determine the content of the documents obtained.

Fact retrieval systems usually require that the information to be retrieved first be placed in a rigid form designed for a particular subject area. This rigid representation for the data, and the corresponding rigid formulation of the retrieval requests, could be produced automatically by a computer which "understands" statements expressed in a form more natural to the human user. Further, if the computer could "understand" information expressed in some general manner, specialized formal representations would be unnecessary.

In order to make a computer serve as a reference librarian, it is not sufficient simply to store a large volume of information. The computer must also have the ability to find and retrieve information in response to flexible descriptive commands. Further, the computer should be able to modify both the information in storage and the requests it is receiving, and it should be able to describe its actions and to request clarifying information. The most useful information retrieval system will be one which can "converse" with its users, to make sure that each request is well-defined and correctly "understood."

2) Mechanical translation: Researchers in the area of mechanical translation of natural language have been disappointed to discover how difficult their task is. First word-to-word translations, and then word-to-word translations coupled with grammatical analysis, rearrangement, and context-dependent restrictions, have proven inadequate for achieving good translations. The vital feature missing from present computer translating systems is the ability of human translators to "understand" what they read in one language, and then "say the same thing" in another. The SIR computer system can store facts, make logical deductions, answer questions, and exhibit other features of human conversational behavior, and therefore appears to have some such "understanding" ability. The mechanisms which help it to "understand" are likely to help also in solving the mechanical translation problem.

3) General computer applications: During the past decade there has been tremendous growth in the amount of computer utilization and in the variety of computer applications. However, before each new problem can be tackled by a computer someone must perform the arduous

task of "programming" a solution, i.e., encoding the problem into a form acceptable to a computer.

Various "problem-oriented" computer languages have been developed to ease this encoding problem. Unfortunately, such languages are useful only when programs ("compilers" or "interpreters") are available to translate automatically from the problem-oriented language to the basic "order-code" of the computer. At present all such problem-oriented languages are very rigid systems. This means that the problem domain must be one which lends itself to rigorous, complete, formal definition, e.g., algebraic manipulations, accounting procedures, or machine tool operations.

Many interesting problems are not sufficiently well defined or clearly understood to be expressed in any of the conventional computer programming languages. Still, people are able to describe these problems to each other and to assist each other in making the problems more precise and in solving them. In order to utilize the high speed and large memory capacities of computers while working on such ill-defined problems, people need some useful way to communicate incomplete information to the computer; some way which will make the computer "aware" of facts and enable it to "understand" the nature of the problems which are described to it. SIR is a prototype of a computer system which captures some measure of the "meaning" of the information presented to it, and can act upon its stored body of knowledge in an "intelligent" manner.

Chapter II: Semantic Information Retrieval Systems

The word "semantic" is used in the title of this paper for two reasons: First, the actual information extracted from text and stored by the program is intended to approximate the linguistic "semantic content" or "meaning" of the material. Second, the computer representation of information used in SIR (Chapter III.B) is derived from the "semantic" model structures of formal mathematical logic. "Information retrieval" refers to the fact that the systems discussed operate on collections of statements, retrieving facts in response to questions. Question-answering was chosen because it is a straight-forward context in which to experiment with the understanding and communicative ability of a computer.

The SIR system utilizes results from two major research areas: the study of the semantics of natural language, and the study of previously developed computer programming techniques for solving various specific question-answering problems.

A. Semantics.

Semantics is generally studied from one of two viewpoints: pure and descriptive. Pure semantics, as studied by Carnap (5), deals with the properties of artificially constructed formal systems (which may or may not have analogues in the real world), with respect to rules for sentence formation and designation of formal models and truth values. I shall rather be concerned with descriptive semantics, an empirical search for rules governing truth and meaningfulness of sentences in natural language.

1) Semantics and meaning: When discussing meaning, one quickly encounters difficulties in having to use words with which to discuss the meaning of words, especially that of the word "meaning." Therefore one finds it difficult to distinguish between object-language and meta-language. A common device is to define "meaning" in a very specialized sense, or to deny that it can be defined at all. Quine, tongue in cheek, recognizes this difficulty in the following paragraph: (33)

"One must remember that an expression's meaning (if we are to admit such things as meanings) is not to be confused with the object, if any, that the expression designates. Sentences do not designate at all..., though words in them may; sentences are simply not singular terms. But sentences still have meanings (if we admit such things as meanings); and the meaning of an eternal sentence is the object designated by the singular term found by bracketing the sentence. That singular term will have a meaning in turn (if we are prodigal enough with meanings), but it will presumably be something further. Under this approach the meaning (if such there be) of the non-eternal sentence 'The door is open' is not a proposition..."

Quine continues that the elusive meaning of "The door is open" is some complete intuitive set of circumstances surrounding a particular occasion on which the statement "The door is open" was uttered. Clearly this kind of concept does not lend itself to computer usage. In order to construct a computer system which behaves as if it understands the meaning of a statement, one must find specific words and relations which can be represented within the computer's memory, yet which somehow capture the significance of the statement they represent.

Ziff (47) is more precise in making the following distinction: words may have meaning, but not significance; utterances (phrases, sentences) may have significance but not meaning. However, he states that an analysis of the <u>significance</u> of a whole utterance cannot be completed without an analysis of the <u>meanings</u> of the words in the utterance. I find Ziff's distinction between word meaning and utterance significance a useful distinction, although the terminology is poor since both concepts contribute to what is commonly called "meaning." Since Ziff does not present any further explanation or representation of "meaning" and "significance," let us proceed to a similar but more complete discussion by Ullmann (44).

Ullmann considers a word as the smallest significant unit with isolated "content," whereas phrases and sentences express <u>relations</u> between the things which are symbolized by individual words. Here "meaning" is defined as "a reciprocal relationship between the name and the sense, which enables the one to call up the other." By "sense" is meant the thought or reference to an object or association which is represented by the word. Note that meaning here relates word with thought about object, not necessarily with object itself. Now, "thought about object" is too vague an idea for computer formalization. However, we <u>can</u> work with a verbalization of a thought; namely, the <u>words</u> which name objects and features associated with the thought.

We may consider the meaning of a word which names an object or class of objects to be either the thing named or, after Ullmann, the most common thoughts people have in connection with the thing named.

In either case, in the SIR system I approximate the meaning of the word by building up, in the computer, a description of the object or class. This description, itself composed of words, presents properties of the described entity and names other objects and classes to which that entity is related. The meaning of an utterance can then be represented in a natural way by particular entries in the descriptions of the objects named in the utterance.

Walpole (45) points out similarly that a word may be defined (*i.e.*, the meaning of a word may be explained) by any kind of association, connection, or characteristic, and these features of a word are usually described verbally. Thus such features can be part of the computer's description of the word being defined.

"Words do not live in isolation in a language system. They enter into all kinds of groupings held together by a complex, unstable and highly subjective network of associations; associations between the names and the senses, associations based on similarity or some other relation. It is by their effects that these associative connections make themselves felt;.... The sum total of these associative networks is the vocabulary." (45)

SIR uses an approximation to those associative networks as its basic data store.

Walpole also notes that some word relationships, such as part to whole, or class to subclass, determine partial orderings of large classes of nouns and thus can be represented by tree structures. This fact leads to certain search procedures which are useful in our computer system. However, the class of abstract nouns ("fictions"), which do not name any object in any specific sense-experience, do not lend themselves to such ordering, and hence are omitted from early versions of computer representations for semantic information.

2) <u>Grammar and meaning</u>: Thus far I have discussed meaning (semantics) while ignoring the grammar (syntax) of language. However, grammar is important since I would like the computer program to take advantage of whatever useful information is available in the grammatical structure of its input. Also, at least one school of thought (discussed in (3) below) holds that syntactic analysis is an adequate method for obtaining semantic classification. Therefore let us consider the nature of grammar.

A "grammar" is usually defined as a set of rules defining which strings of alphabetic characters are "sentences" of the language and which are not. Deriving a grammar for a natural language is an empirical process, since the ultimate test of whether a statement is grammatical or not is to ask a native speaker. Considering only the functions of words in sentences (their "parts of speech") but not their meanings in any sense, Chomsky (9) develops various kinds of English grammars. <u>Phrase structure</u> is a simple concept and works for a small part of the language, but is frequently inadequate. <u>Transformational grammar</u> schemes are probably adequate, but are complicated and difficult to complete or test.

Although syntactic procedures are generally supposed to ignore meaning, the boundary between syntactics and semantics is hazy. For example, some linguists classify the so-called "mass nouns" (e.g. "water") as a separate grammatical group since they do not take the article. However, the distinction between "I want meat" and "I want a steak" seems to be basically a semantic one.

Ziff defines meaningfulness in terms of rigidity of grammatical structure. Words which are necessary in a particular grammatical configuration, such as frequent occurrences of "to," "do," "the," and the like, are said to have no meaning. On the other hand, words which could be replaced by a large number of alternatives within a given grammatical context are considered very meaningful. Simmons (38) makes this distinction between function words and content words even more sharp, as we shall see later (Paragraph C.3). I have used these ideas to the extent that only words which are names of objects or classes, or of properties of objects or classes, appear in the internal representation used in SIR. The frequently-occurring "meaningless" words of Ziff are used as indicators of relations between other "meaningful" words appearing in the same sentence. (See Section IV.B).

3) Formalizing meaning: The intelligent computer has to understand and remember the meaning of what it is told; therefore it needs some precise internal representation for these meanings. Let us now examine some of the formal representations of meaning which have been proposed, and see which ideas from those representations might be useful in a computer representation.

One way to deal with the problem of semantics is to avoid it by translating ordinary language into a formal system which could be handled syntactically (1). Thus far, attempts to formally encode all of natural English seem to introduce a mass of detailed notation which obscures the real problem, for the problem of representing meaning must be solved in order to develop a good translation scheme. At first view Freudenthal's LINCOS (15) may seem like a formal system for describing human behavior. Actually the LINCOS system is not practical since it assumes far greater abilities for inductive inference of rules and situations on the part of the receiver than is expected of the usual language student.

Another approach, used, for example, by Klein (19), is to increase the number and kinds of categories in the usual syntactic analysis systems until the semantic properties are automatically included. Although some of the results are promising, it seems to me this approach will eventually obtain the same ultimate system of word associations as can be approached more simply by considering and representing directly the "meaningful" relations between words.

Quillian (32) attempts to represent the semantic content of words as sets of "concepts," which can be combined to represent the meanings of phrases and sentences. With the basic premise that learning a new word involves measuring its values on a set of basic scales, he is trying to build up a repertoire of suitable coordinate scales. Each word is represented by a set of values which are generally intuitive, unidimensional coordinates such as length, time, and hue. Quillian also permits defining words in terms of predefined words as coordinates. My feeling is that the relations <u>between</u> words are more important than the conceptual meaning of <u>individual</u> words, and therefore a simpler approach which ignores "basic" meanings would be more immediately fruitful.

Sommers (42) is more concerned with permissible word combinations than with the meanings of individual words. He first describes a hierarchy of sentence types: 1) Ungrammatical; 2) Grammatical but nonsense; 3) Sensible but false; 4) True. He then argues that the crucial semantic distinction lies between the grammatical declarative sentences which are nonsense, and those which are significant (but may be true or false). Any pair of monadic predicates P_1, P_2 are said by Sommers to have a sense value $U(P_1,P_2)$ if there exists any significant sentence conjoining them. Otherwise they have value $\sim U=N(P_1,P_2)$. The U-relation is symmetric and is preserved under certain logical operations on its arguments, but it is not transitive. A stronger relation $Q \leftarrow P$ is true if "of (what is) P, it can be significantly said that it is Q...*e.g.*, P=Prime minister, Q=quick." This permits the arrangement of these "monadic predicates" into a simple tree, where all words in the same meaning class, *e.g.* all colors, or all words describing weight, occupy the same node.

My main objection to this work is in where the important distinctions lie. Sommers would argue that "The idea is always green" is nonsense, but "The yellow sky is always green" is sensible (since sky may have color, "The sky is blue" and "The sky is not blue" are significant), although false. Note that "Ideas cannot be green" would be considered nonsense rather than true, by Sommers. I feel the distinction between "nonsense" and "sensible but not true of the real world" is not precise enough to be a basis for a computer representation of a semantic system. SIR is concerned with deductions of consequences from a given body of statements, rather than judgements of "nonsense" or "sensible."

In summary, many schemes have been developed in the literature for formally describing the semantic properties of language. Some of these were described above. Most of the schemes are vague, and although Klein's and Quillian's, among others, are being programmed for computers, none of the presently available semantic systems have been developed to the point where they could provide a useful basis for computer "understanding." However, I have used some of the ideas from the above systems in developing SIR. The idea of representing meaning by word associations is particularly important for the information representation used in SIR.

B. Models:

The SIR system uses a special data structure which I call the "model." The program refers to this "model" whenever it must store or retrieve semantic information. The purpose of this section is to explain what I mean by the term "model" in general, and to define the SIR model in particular.

1) Definition: The term "model" has been grossly overworked, and it does not seem to have any generally agreed-upon definition (18). For purposes of this paper, I present the following definition:

A model for an entity x has the following properties:

a. Certain features of the model correspond in some well-defined way to certain features of x.

b. Changes in the model represent, in some well-defined way, corresponding changes in x.

c. There is some distinct advantage to studying the model and effects of changes upon it in order to learn about x, rather than studying x directly.

x may be any of a wide class of entities, such as an object, a statement in English, or a mathematical concept.

2) Examples of models:

i) A small-scale wind-tunnel test-section for part of an airplane is a model for the actual part because aerodynamicists understand how air flow around the test-section is related to air flow around an actual airplane part (whose shape corresponds to the shape of the test-section in a well-defined way). An obvious advantage of such a model is its convenient size.

ii) A verbal statement of a plane geometry problem usually includes statements about line segments, connections, shapes, etc. The usual model is a pencil or chalk diagram which has the geometric features described in the statement. The advantage of the model is that it is conceptually easier for people to interpret geometric relationships from a diagram than from a verbal statement, which is really an encoding of the geometric information into a linear string of words.

iii) Problem solving ability in human beings has been modeled by a computer program developed by Newell, Shaw and Simon (28). The model can be improved by modifying the program so that its external behavior corresponds more closely to the behavior of people working on the same problems. The advantage of this model for behavior is that its internal workings are observable, and hence provide a hypothesis for the corresponding mechanisms involved at the information-processing level in human problem-solving.

iv) Logicians develop and study formal systems. Occasionally these have no significance other than their syntactic structures. Sometimes, however, systems are developed in order to study the properties of external (usually mathematical) relationships. On these occasions one says that statements in the formal system correspond "under standard interpretation" to facts about the relationships. The model for such a formal (syntactic) system usually consists of sets of objects which satisfy our intuitive notions of the "meaning" of the original relationships, yet whose properties correspond to certain features of the syntactic statements. Thus one may study the abstract formal system by manipulating a model which has intuitive significance. Semantics, in mathematical logic, refers to the study of such models (6).

There may not always be a clear-cut distinction between entities which are models and those which are not really representations of something else. For example, Newell, Shaw, and Simon's problem-solving program discussed in (iii) above is truly a model, in the sense defined earlier, only insofar as it is intended to represent human behavior. Otherwise the program would have to be treated just on its merits as an independent problem-solving machine.

3) Question-answering model: In designing a question-answering system one is concerned with providing a store of information, or a mechanism for developing such a store, and a procedure for extracting appropriate information from that store when presented with a question. The store may be built up on the basis of information presented in the form of simple declarative English sentences, as it is in SIR, or it may be a prepared data structure. In either case, it generally contains information which people would normally communicate to each other in English sentences. I consider the store of information which is the basis of any question-answering system as a model for any set of English sentences which contains the same information. Of course, "information contained" refers here to the semantic content, not the number of information-theoretic bits. Note that, due to the present vague state of semantic analysis in natural language, the most effective

way of discovering this information content of a question-answering system's store of information is to ask the system some questions -- and make subjective inferences from its performance.

The information store of a system is a model for a set of English sentences because the information which can be extracted from the store corresponds in a well-defined way to, and in fact should be identical to, at least some of the information available in the sentences. The principal advantage of such a model is that it is easier to identify and extract desired information from the model than it would be from the complete English sentences. Question-answering systems have been developed which use various kinds of models and which have achieved varying degrees of success. The best-known examples of such systems are discussed in the following section. The structure of the model used in my new question-answering system is discussed in Chapter III of this paper.

C. Some Existing Question-Answering Systems.

Several computer programs have been written whose aims and results are somewhat related to those of SIR. None of these "question-answering" systems uses a model for storing arbitrary semantic information; and none of them deal with the same general kind of subject matter as SIR. However, each of these systems has certain interesting features, some of which have influenced the design of SIR.

1) "Baseball: An Automatic Question-Answerer." (17) This program, written in the IPL-V (25) programming language, answers most reasonable verbal English questions about a set of baseball games. Example:

input: "How many teams played in 8 places in July?"

output: MONTH = JULY

PLACE (NO. OF) = 8

TEAM (NO. OF) = ?: YANKEES, TIGERS, RED SOX.

The stored information (model) consists of a list-structure containing all the relevant baseball game results arranged according to a pre-selected hierarchical format. There is no provision for automatically modifying this model. Each question is translated into a specification-list with the desired information represented by blanks. This specification-list is then matched against the model, the blanks filled, and the entire final specification-list printed out. No attempt is made to respond in grammatical English.

The bulk of the program is devoted to the task of translating a question sentence into a specification-list. This requires looking up words in a dictionary, identifying idioms, performing grammatical analysis, resolving ambiguities, etc. The dictionary consists of a set of entries for each word, such as its part of speech, whether the word is part of an idiom, and its "meaning." "Meaning," which only appears for certain words, refers to a canonical translation of the word within the context of the program; e.g., the meaning of "who" is "Team = ?" Thus the specialized nature of the subject matter enables simple, *ad hoc* procedures to solve what would otherwise be very difficult problems. The model consists of a fixed structure of information arranged to facilitate the process of filling blanks in specification-lists.

The "Baseball" system gives the illusion of intelligent behavior because it can respond to a wide variety of English question forms. However, a limited amount of information about a specific subject must be pre-arranged in a fixed data structure, and the data must lend itself to hierarchical ordering. Such a scheme cannot be generalized conveniently to handle the larger variety of information which is necessary for a truly "intelligent" system.

2) Phillips' "Question-Answering Routine." (31) This program, written in the LISP programming language, (23) can correctly answer certain simple English questions on the basis of a corpus of simple English sentences.

Example:

input: ((AT SCHOOL JOHNNY MEETS THE TEACHER)

(THE TEACHER READS BOOKS IN THE CLASSROOM))

(WHERE DOES THE TEACHER READ BOOKS)

output: (IN THE CLASSROOM)

The model for a sentence is a list of up to five elements: subject, verb, object, place, and time. This model is constructed for each sentence in the corpus, and for the question (where a special symbol in the question-list identifies the unknown item). The question-list is matched against each sentence-list and, if an appropriately matching sentence is found, the correct reply is extracted from the corresponding sentence in the original corpus.

This is a primitive system in several obvious respects: any information in a sentence other than the five "basic" elements, and any sentence which cannot be analyzed, is ignored; the words in the question must be exactly the same as those in a corpus sentence; a question must be answerable on the basis of a single sentence from the corpus; and the model for the entire corpus must be searched linearly for the answer to each question. However, the idea of a model which is created and extended automatically as new sentences are added, and which serves as an intermediary form to assist in finding answers to questions, is an essential feature of an intelligent, human-like system -- and is the important contribution of Phillip's work.

3) "SYNTHEX." (38) This program, written in the JOVIAL programming language (37), can answer a wide variety of questions about information contained in a large corpus of simple natural English such as the Golden Book Encyclopedia. Example:

input: "What do birds eat?"

(somewhere in the encyclopedia): "Worms are eaten by birds."

output: "Birds eat worms."

The program classifies all words as either function words, which have structural (syntactic) significance (e.g., the, is, do, what), and content words, which have semantic significance (in practice, content words are any words which have not been chosen as function words).

Initially the corpus (the encyclopedia) is indexed with respect to all occurrences of all content words. This index occupies about the same amount of space as the corpus itself. When a question is asked, the system selects those sentences from the corpus which have the greatest number of content words in common with the question. At this point elaborate grammatical analyses are used to determine whether any of the selected sentences provide an answer to the question.

This system doesn't use a model at all; the complete corpus is kept in its original form and referred to, when necessary, through the use of an index. Since the information is not pre-processed into a more usable form, the grammatical analysis required at the time the question is answered is quite complex. Recent related work by Klein (19) indicated that some of the rules of the grammar can be developed automatically from the corpus, and information from several sentences may be combined by use of syntactic methods to help answer questions.

My feeling is that the word-relations being developed by these "dependency grammar" methods can be discovered more easily by means of <u>semantic</u> analysis, and they would then be more intuitively meaningful. A model based on such semantic relations would significantly simplify the question-answering procedure. SIR illustrates the feasibility of directly storing and using semantic relations.

4) <u>Lindsay's "SAD-SAM: Sentence Appraiser and Diagrammer, and Semantic Analyzing Machine."</u> (21) This program, written in the IPL-V (26) programming language, accepts as input any sentence in Basic

English (30), extracts from it any information concerning kinship, and adds this information to a "family tree." Example:

input: "John, Mary's brother, went home."

effect: John and Mary are assigned a common set of parents -- i.e., they are represented as descendants of a common node in the family tree. The grammar is sufficient to handle a considerable portion of natural English in recognizing family relationships. Although the author does not consider question-answering in detail, it is clear that the family relation information is immediately available in the tree model and specific requests could be answered almost trivially.

This system illustrates the effectiveness of a model designed for a very specific task. Lindsay decided in advance that only family relationships were of interest, and observed that there is a natural model for family relationships. Then whatever relevant information was received was processed into this model, leaving practically nothing to be done at question-answering time.

Unfortunately, different forms of "natural" models are needed for different kinds of information. In a more general system, it might be possible to use the best available model to represent information for each subject area --e.g., trees for family relations, Cartesian coordinates for spatial relations, perhaps just the original text in areas for which there is no obviously better representation; but that would be a confused system with tremendous organizational problems. The SIR system is based on a single model which captures some of the advantages of various specific models while permitting uniform processing procedures and permitting the storage and retrieval of arbitrary facts which arise in human conversation.

5) <u>Darlington's program for the translation of restricted English into the notation of symbolic logic (12)</u>: This program, written in the COMIT (34) programming language, translates certain English riddles into a logical form which may then be tested for validity by another program, written by the same author, which applies the Davis-Putnam proof procedure (13) for statements in the propositional calculus.

Example:

<u>input</u>: "If the butler was present, then the butler would have been seen, and if the butler was seen, then the butler would have been questioned. If the butler had been questioned, then the butler would have replied, and if the butler had replied, then the butler would have been heard. The butler was not heard. If the butler was neither seen nor heard, then the butler must have been on duty, and if the butler was on duty, then the butler must have been present. Therefore the butler was questioned."

<u>output</u>: $[[L \Rightarrow M] \wedge [M \Rightarrow N] \wedge [N \Rightarrow P] \wedge [P \Rightarrow Q] \wedge \sim Q \wedge [\sim M \wedge \sim Q] \Rightarrow R] \wedge [R \Rightarrow L]] \Rightarrow N]$

The input is typical of a type of problem which appears in elementary logic texts. It has been pre-edited to perform certain clarifications including removal of most pronouns and insertion of necessary marker words such as "then." The program translates this input, by means of dictionary references and grammatical analysis, into the model, which is a statement in mathematical logic having the same truth-value as the original English statement. The "question" in these problems is understood to be, "Is this argument valid (<u>i.e.</u>, necessarily true)?", and the answer can be obtained by applying established methods to the logical model.

As in Lindsay's kinship system (4) above, Darlington's program takes advantage of a model ideally suited to the type of problem involved and advance knowledge of the only possible question. If one considers the possibility of questions such as, "What was the occupation of the suspect who was questioned?," or "What was done to the butler?" then the complicated process of translating the corpus into logical terms would not be of any aid in finding answers. Only a small part of the information needed for intelligent behavior can be expressed in the propositional calculus. As will be discussed in Chapter VI, even a version of the quantificational calculus is not sufficient to formalize the conversational ability of SIR; a procedural language is also necessary.

6) <u>Bennett's computer program for word relations.(3):</u> This program, written in the COMIT programming language, will accept information and answer questions framed in a small number of fixed formats. Example:

<u>input</u>: DOG IS ALWAYS MAMMAL.

MAMMAL IS ALWAYS ANIMAL.

WHAT IS ALWAYS ANIMAL Q.

<u>output</u>: MAMMAL IS ALWAYS ANIMAL.

The input sentences must be in one of five formats (<u>e.g.</u>, "<u>X</u> IS ALWAYS <u>Y</u>," "<u>X</u> MAY BE <u>Y</u>," etc.), and only one occurrence of each format may be held true at one time for any one item <u>X</u>. This input information is translated into the model, which has associated with every item <u>X</u> each corresponding item <u>Y</u> and an identifying number for the format which set up the correspondence. (The model actually consists of linear strings of tagged entries, as is required by the COMIT language.) Similarly there is a small number of allowable question-formats, each associated with one of the input-formats and resulting in a particular class of entries being retrieved from the model.

The major feature of this system, which is also the basic feature of SIR, is that the information kept in the model identifies particular kinds of semantic relations between particular words. Questions are analyzed with respect to, and answered by referring to the model for information about, these same relations. Principal shortcomings of Bennett's system, which I have overcome in SIR, include the following:

1) Relations are identified with particular formats rather than with their intended interpretations.

2) Logical implications based on the meanings of the relations are ignored.

3) Interactions between different relations are ignored.

4) Its string representation makes processing the model more difficult than necessary.

5) The user must know the form and content of the model in order to make changes to it.

In summary, several computer question-answering systems have been developed to solve special problems or illustrate special abilities. None of them constitute a direct approach to providing intelligent "understanding" behavior for the computer. Although various forms of models are used in the existing systems, none represent semantic relations in an intuitive, <u>general</u>, and useable way. The SIR model described in the next chapter provides the basis for a system which is more powerful than any developed thus far. The system based on this model can store and retrieve information about arbitrary subjects, make logical deductions, account for interactions between stored relations, resolve certain ambiguities, and perform other tasks which are necessary prerequisites for an understanding machine.

Chapter III: Representations for Semantic Information

The SIR model is the collection of data which the SIR programs can refer to in the course of question-answering. It is a dynamic model, in the sense that new information can cause automatic additions or changes to the data. In addition, it is a semantic model, in the sense that the data are organized in a structure which represents the meanings of the English sentences upon which the model is based. The purpose of this chapter is to describe this semantic organization, which is reponsible for convenient accessibility of relevant information and therefore for efficient question-answering.

Many kinds of "semantic" models are possible. The precise form of the SIR model evolved from studies of possible word-association models and of the semantic systems of mathematical logic. Its implementation was influenced by the features of available computer programming languages. It is only capable of representing a particular group of semantic relations. These factors are discussed in the following paragraphs. Chapter VI will present a proposal for future expansion and formalization of this model and of its associated programs.

A. Symbol-Manipulating Computer Languages (4)

Programming the SIR system, or any other elaborate question-answering system, would have been almost impossible if not for the availability of symbol-manipulating computer languages. By taking care of much of the necessary encoding and bookkeeping, these languages permit a programmer to concentrate on the more significant aspects of organization and representation necessary for problem-solving. Since the choice of a symbol-manipulating language was an important step in the development of SIR, it seems worth-while to discuss this class of languages in some detail.

Historically, the data used in computers have been numerical, in the form of either numbers or fixed-size vectors and arrays of numbers. Question-answering and other areas of recent computer research require the use of symbolic as well as numeric data, and it is frequently desirable to transmit information by means of the relational structure as well as the symbolic content of the data. The "symbol-manipulating" or "list-processing" computer languages have been developed to handle these special processing needs. An important feature of these languages is that computer memory space for data structures need not be preassigned; storage for each structure is allocated automatically as it is needed. Thus a symbol-manipulating language gives a programmer a powerful set of tools for describing processes which create, modify, search, or otherwise operate on arbitrary amounts of symbolic data without being concerned with the inherent limitations or basic numerical operations of the computer being used.

The most widely used symbol-manipulating computer languages are IPL (25), COMIT (35), and LISP (23).* IPL, used in the "Baseball" and "SAD-SAM" question-answering systems described in the previous chapter,

---

* See reference (4) for definitions of list-processing terms and more detailed descriptions and comparisons of these languages.

is one of the oldest symbol-manipulating languages. The basic units of data used in IPL are list structures* composed of IPL symbols. An IPL program describes symbol manipulation at a very basic level, leaving the programmer with the problems of keeping track of storage used, symbols assigned, etc. On the other hand, it is quite easy in IPL to build up elaborate programs out of simpler processes and to manipulate arbitrarily complex list structures.

COMIT was originally designed to be a convenient system in which to process natural language, and was used in two of the question-answering systems described above. Although COMIT is a general purpose symbol manipulation system, it is best suited to problems involving string* manipulation; i.e., problems in which the data can be represented in the form of strings of symbols without introducing undue complication into the processing algorithms. The COMIT system provides a simple yet powerful formalism for describing string manipulations. This formalism can be extremely useful for describing procedures, such as parsing, which operate on sentences of natural language.

LISP, the language used in one of the above question-answerers and the one chosen for programming SIR, was originally designed to be a formalism useful for studying the mathematical properties of functions of symbolic expressions as well as useful in a practical programming system. LISP programs consist of <u>functions</u>, rather than sequences of instructions

* See reference (4) for definitions of list-processing terms and more detailed descriptions and comparisons of these languages.

or descriptions of data forms. These functions map symbolic expressions into symbolic expressions; the basic form of a LISP symbolic expression is a binary tree* which can easily be used to represent list structures when necessary. The organization of LISP programs into functions enables one to describe elaborate recursive tree-searching and list-structure-building operations simply and concisely. Reasons for choosing LISP as the language for programming SIR include the following:

1) Unlike IPL, LISP offers several significant programming conveniences such as the use of mnemonic symbols and the automatic maintenances of available storage.

2) Unlike COMIT, complex trees and list structures -- which frequently arise in the chosen representation for the model (see section D) -- can be represented directly as LISP data.

3) The LISP formalism is particularly well suited for describing the recursive tree-searching procedures which are an important part of the system (see Chapter V).

In an earlier version of SIR, COMIT was used as a pre-processor to translate from English sentences into a function form better suited for LISP input. However, since the simple format-matching input procedures finally chosen (see Chapter IV) could just as easily be handled in LISP, the problems of a hybrid system were avoided by converting everything to the LISP language.

B. Word Association Models

The variety of existing question-answering systems discussed in the previous chapter demonstrates that many different kinds of models for

* See reference (4) for definitions of list-processing terms and more detailed descriptions and comparisons of these languages.

representing the information in English text are possible. One can develop question-answering systems which vary widely in approach. At one extreme are systems, e.g., Lindsay's kinship program, which immediately process the text into a form from which anticipated questions can be answered trivially, but which thereby ignore much of the information in the input. At the other extreme are systems, e.g., the SYNTHEX system, which simply store the raw text and perform all necessary computations after each question is received, thereby becoming embroiled in complex grammatical analysis.

I feel that a system which is capable of intelligent, human-like behavior must lie between these two extremes. Accordingly, the design requirements for the model in SIR included the following:

i) The model organization should be general enough to be useful in a wide variety of subject areas, yet the stored information should be specific enough to be of real assistance in the question-answering process.

ii) The effort involved in the question-answering procedure should be divided between the job of encoding input into the model and the job of retrieving answers from the model. Neither job must be prohibitively complicated or time-consuming.

Models based upon words and word-associations are the best candidates for meeting these requirements.

<u>Words</u> are the basic symbols in most natural languages. Certain words, usually verbs and prepositions, denote <u>relationships</u> between real objects. In the SIR model I shall use words themselves to represent the objects or classes denoted by the words, and specific kinds of associations between words to represent relations between those objects or classes.

Before describing the kinds of associations actually used in the SIR model, let us consider a simpler word-association model structured solely by class-inclusion ("$\subset$") and class-membership ("ϵ") relations. This model, which was considered early in this investigation, has certain possible applications, but also has significant drawbacks which prevent its use in SIR. It is structured as follows: Let $\underline{X}$ and $\underline{Y}$ be words which denote the objects or classes represented by $\underline{x}$ and $\underline{y}$, respectively. All such words are arranged in a tree, i.e., partially ordered, according to the following rule: $X < Y$ if either $x \subset y$ or $x \epsilon y$. In addition to this primary ordering, various kinds of secondary associations can be indicated by special additional links. Similarly, some verbs can be partially ordered. For example, if $\underline{x}$ and $\underline{y}$ denote the subject and object, respectively, of a verb $\underline{\alpha}$ in a sentence $x\alpha y$, we shall order verbs by the criterion: $\alpha < \beta$ if, for all objects $\underline{x}$ and $\underline{y}$, $x\alpha y$ implies $x\beta y$. For intransitive verbs, the criterion is $\alpha < \beta$ if $x\alpha$ implies $x\beta$. Fig. 2 shows such trees for some words from a first-grade reader (29). The parenthesized words were not in the vocabulary of the text, but are included to motivate the organization of the tree.

Having defined the tree of nouns and the tree of verbs, I must now complete the model by defining connections between these two trees. Although a formal notation for such cross-links could be defined, for present purposes I shall simply give the following examples of statements describing cross-linkages (with respect to the node-labeling in Fig. 2):

i) Any noun below node 1 is a suitable subject for any verb below node 1'.

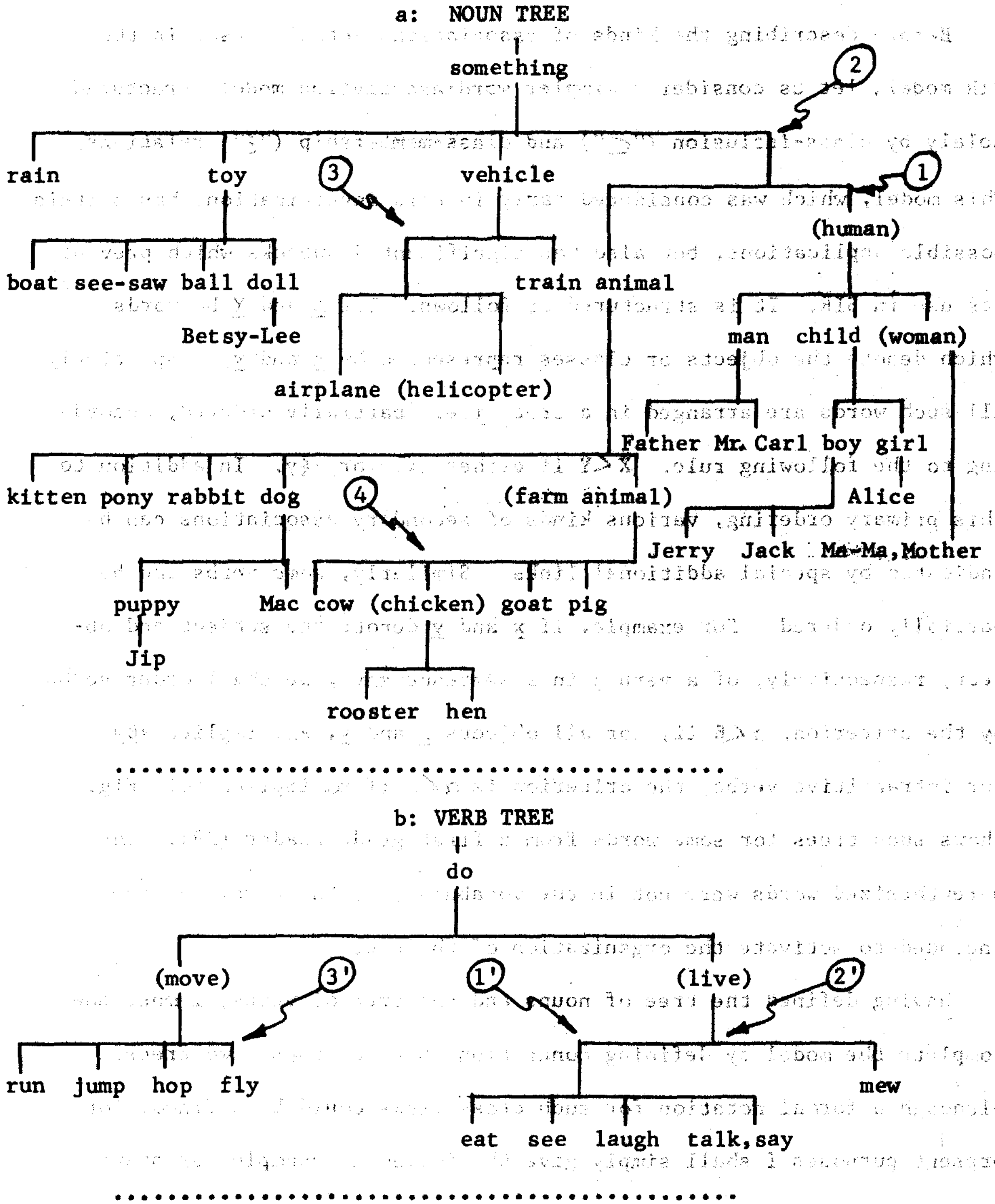

FIGURE 2: A WORD ASSOCIATION MODEL

ii) "Any noun below node 2 is a suitable subject for any verb above node 2'.

iii) Only nouns below nodes 3 or 4 may be subjects for verbs below node 3".

The complete model, composed of tree structures and statements about their possible connections, is a representation for the class of all possible events. In other words, it represents the computer's knowledge of the world. We now have a mechanism for testing the "coherence" or "meaningfulness" of new samples of text. As information is fed into a system which uses this model, the program would simply have to insert a "thread" of special connections into the model. The thread would distinguish those events which actually happened from those which are just "conceivable" to the computer. Questions about the input statements could then be answered by referring to the model to see which way the thread passed. Such a model would be useful in a pragmatic system such as Abelson's (7), to test the credibility of what it is told. It could identify sources of its factual knowledge by their threads, and compare the reliabilities of the various sources.

Unfortunately, the model described has several drawbacks which prevent its use in a general semantic information retrieval system. It is extremely difficut to construct a useful model of the form described, for a significant amount of information; writing a program which would add information to the model __automatically__ is out of the question. The "$\subset$" and "ϵ" relations are not sufficient to describe many useful groupings of nouns, but the introduction of a few additional relations would confuse the structural organization of the model and force the cross-link statements to be much more complicated. The verb groupings, in order to be useful, must be carefully selected according to the ill-defined restriction that the resulting configuration allow simple and useful cross-link statements. This may not always be possible, and certainly becomes more difficult as the number of relations considered increases.

The model used in SIR is a word-association model similar in some respects to the one just described. However, the words are linked in a general manner so that no particular relations are more significant than others. The model is constructed, on the basis of input sentences, completely automatically. Descriptions of the behavior of particular relations, which roughly correspond to the cross-link statements in the above system, are programmed into SIR rather than being part of the model. Section D below describes the actual model used in SIR.

C. Semantics and Logic.

The structure of the SIR model was partly motivated by the structure of models in mathematical logic. These logical models represent the "meanings" of logical statements, and thereby help the mathematician "think" about his problems, in the same way that the SIR model is supposed to represent the "meaning" of English input, and thereby help the program obtain answers to questions. Let us take a more detailed look at logical models.

The "semantics" of mathematical logic is the study of models for logical systems (6). Such a model consists of a set of individuals (corresponding to the domain of the logical variables), and, for each logical predicate or relation, a set of ordered n-tuples of individuals.

A relation is true of certain individuals if and only if, in the model, the ordered n-tuple of those individuals is an element of the set corresponding to the relation. For example, a model for a logical system dealing with the natural ordering of the integers might have as its model the set of integers (as the domain of individual variables) and a set of ordered pairs of integers corresponding to the "<" (less-than) relation. This latter set would contain all pairs (a, b) for which integer a is truly less than integer b, i.e., for which the statement a<b is true.

These semantic models are particularly useful in logic for studying certain properties, such as consistency and completeness, of the associated formal systems. They are not generally as useful as aids in proving particular theorems, or studying the possible interactions between various relations. The SIR model organization must be better suited to these latter problems, which are of major interest in developing a question-answering system.

The idea of representing a relation by a set of ordered n-tuples is a good starting point for a question-answering system model. However, certain modifications are necessary. Since we are interested in conversational ability in the computer, the "relations" in our model should represent concepts which commonly occur in human conversation, such as set-inclusion and spatial relationships, rather than abstract mathematical properties. Furthermore, unlike a logical model, the system should have built-in provisions for determining restrictions, extensions, or inconsistencies in the model, based on properties of the relations involved. E.g., if "$\subset$" indicates set-inclusion, and if $a \subset b$ and $b \subset c$ are both in the model, the system should deduce that $a \subset c$ should also be in the model (or, equivalently, that $a \subset c$ is a true statement), from the built-in knowledge that set-inclusion is transitive. Finally, for reasons of computational efficiency, a subject which is never considered in formal logic but is of prime importance in a practical computer system, information about relations must be more easily accessible than it would be if it consisted simply of unordered sets of n-tuples of objects. These considerations led to a choice of the description-list organization for the actual word association model used in SIR and described in Part D below.

Although some ideas were borrowed from logical semantic systems, SIR is not directly dependent upon any formal logical mechanism. Instead, the model and the programs which utilize it were designed according to informal heuristic principles of reasoning, which I believe to be the most convenient ones for a first experimental system for intelligent conversation between machines and human beings. Once a working system has been developed, one can try to extract from it a logical basis for a more advanced system. Such an extension is the subject of Chapter VI.

D. The SIR Model.

The SIR model consists of words associated with each other through particular relations. These associations are represented by "description-list" entries. In this section I shall discuss the description-list structure, the relations used in SIR, and the precise representations for those relations.

1) Description-lists: The model in SIR is based largely upon the use of description-lists. A description-list is a sequence of pairs of elements, and the entire list is associated with a particular object. The first element of each pair is the name of an attribute applicable to a class of objects, and the second element of the pair is the value of that attribute for the object described. For example, if the object is the number "3", its description-list might contain the following sequence of attributes (underlined) and associated values:

SUCCESSOR, 4, ODD, YES, SHAPE, CURVY, ...

The fact that "3" is an odd number could have been indicated simply by the presence of the attribute "ODD," with any associated value -- or no value at all, provided the system using the description-lists is capable of recognizing such a "flag," i.e., valueless attribute.

The class of "cats" might be described by the list:

SOUND, MEW, COLOR, (BLACK, WHITE, YELLOW, BROWN), LEGGEDNESS, 4, ...

Note that, since the color of cats is not unique, the value associated with COLOR is a list of possible cat colors. Its enclosure in parentheses indicates that the entire list of colors is a single element of the description-list.

I can illustrate the way description-lists may be used by considering their place in the IPL (25) programming system. By convention, every IPL data list has an associated description-list. The attributes on IPL description-lists are IPL symbols, and the values are symbols which may name arbitrarily complex IPL list structures. Basic IPL operations can add pairs to description-lists; others retrieve the second element of a pair (a value) on the description-list, given the first element (the attribute) and the name of the main data list. An attribute can only occur once on any one description list, and the order of the attributes on a description-list is ignored. Thus, description-list operations simulate an associative memory containing arbitrary descriptive information for the described object.

The LISP system (23) utilizes "property-lists" which are used in much the same ways as IPL description-lists. In LISP, the described objects are individual words or "atomic symbols," rather than lists. LISP associates with each unique atomic symbol a property-list which is a description-list allowing the use of flags as well as attribute-value pairs. Although originally provided to facilitate the internal operations of the LISP system, property-lists may be searched and modified by the programmer. The model in SIR depends upon the use of property-lists.

2) Model organization and development: The purpose of the model is to assist the computer in understanding and communicating with a person in English sentences. SIR works only with simple sentences which consist of words which denote real objects or classes of objects and words which express particular relationships between the objects and classes. If one considers the objects and classes

as the individual elements in a formal system, then these relationships between objects and classes are analogous to the relations of formal logic (described in C above). "Understanding the meaning" of a sentence is interpreted as the process of recognizing the objects in the sentence and of placing them in a specified relation to one another. The proper relation to use is frequently determined by the verbs and prepositions in the sentence, and the way in which to place the objects into the relation is determined by the form of the sentence. For example, the verb "is" usually determines a set relation. The form "Every x is a y" determines that class x is a subset of class y.

In the computer representation the basic objects, as well as the names of relations, are simply words. The intended interpretation of this representation is as follows: Suppose word x is associated in the model with word y by means of relation R. Then this represents a statement which "means" that the object or class denoted by x is associated with the object or class denoted by y by means of the relation named R.

The procedure for developing the form of the model and the associated storage and retrieval programs was approximately as follows: A single relation -- set inclusion -- was chosen because it is an easy concept to recognize from English text and is also (intuitively) important to the "meaning" of simple sentences. An internal computer representation was then found which ~~adequately represented the relational~~ information, seemed general enough to model many other kinds of relations, and also had connectivity and accessibility properties which make it useful for question-answering. Programs were then developed for recognizing sentences which deal with the given relation by their syntactic forms (see Chapter IV); selecting relevant word tokens from the sentences; and adding to, modifying, or searching the model according to the results of the recognition process. The search programs are designed to "know" the peculiar properties of the relation being searched, *e.g.*, transitivity or reflexivity. Therefore a special set of search programs had to be written for each relation. Each time a new concept or relation was added to the system, the above steps were repeated. That is, the basic model structure was generalized, if necessary; new syntactic recognition forms were introduced, and existing ones modified if any ambiguities had been introduced; and search and response programs for the new relation were written. Search programs designed for relations already available in the system were modified when the old and new relations "interacted"*

The relations included in SIR were chosen because they demonstrate various aspects of the information normally conveyed in human conversation. They were introduced in the following order and for the reasons stated:

a) Set-inclusion, because it is one of the most basic relations of which people are aware.

b) Part-whole relationship, because, although it is significantly

* "Interactions" between relations, and the structure of a modified system which is easier to expand, are discussed in Chapter VI.

different from, it interacts strongly with the set-inclusion relation and has several common properties with it permitting the use of common subroutines.

c) Numeric quantity associated with the part-whole relation, since it is not a new relation but rather consists of special descriptive information which must be carried along with relational information.

d) Set-membership, because it is closely related to set-inclusion but requires attention to properties of individual objects as well as classes.

e) Left-to-right spatial relations, to see how the chosen model works for a different kind of relation for which there is a different, more natural-appearing model.

f) Ownership, since it is quite different from the existing part-whole relation, and yet frequently is specified by the same verb ("to have"). It is therefore a suitable subject for an experiment in resolving ambiguities.

3) Model structure: The basic objects in the model are the words which denote real objects and classes. If an English statement is interpreted by the sentence-form recognition program as asserting that relation R holds between objects or classes named x and y, then this relationship is represented by placing attribute-value pairs on the property-lists of both x and y. Each attribute specifies a relation, and the value of the attribute indicates which other objects are related to the described object by means of the specified relation.

Since in general relations are not symmetric, relation R must be factored into two relations R1 and R2 so that if relation R holds between x and y (in logic terms, if $\langle x,y\rangle \in R$), then one can say that y stands in relation R1 to x and x stands in the inverse relation R2 to y. One may think of R1 and R2 as mappings from individuals into sets such that $\langle x,y\rangle \in R$ if and only if $y \in R1(x)$ and $x \in R2(y)$. For example, if R is the set-inclusion relation, R1 is the subset relation and R2 the superset relation. R1 and R2 may be named by the symbols SUBSET and SUPERSET. In general, the symbols naming R1 and R2 are used as attributes on the property lists of x and y, respectively. Note that if R is a symmetric relation then only one mapping, which may itself be named R, is necessary; for $y \in R(x)$ implies $x \in R(y)$ and vice-versa.

If one and only one object can be in relation R1 to any word x, then the value of attribute R1 of x can be simply the name of that object. In this case I say that a type-1 link exists from x to y following (or, by means of) the attribute R1. An example of the use of type-1 links is in spatial relations, where only one object can be "just-to-the-right" of another. If the system learns that "The lamp is just to the right of the chair," then the attribute-value pair (JRIGHT, LAMP) is added to the property-list of CHAIR, and the inverse relation is indicated by adding the pair (JLEFT, CHAIR) to the property-list of LAMP.

If R holds between x and y and also between x and z, type-1 links are inadequate, since there can only be one value corresponding to a given attribute on a given property list. However, this value may be a list of object-names instead of just a single object-name. In particular, we can make the value of R1 a list of the objects related to x by relation R. For example, in the set-inclusion relation we may learn independently that every boy is a person, every girl is a person, and every MIT-student is a person. The value of the attribute SUBSET on the property-list of PERSON would then be the list (BOY, GIRL, MIT-STUDENT). This type of linkage is called a type-2 link.

Occasionally descriptive information pertinent to a particular occurrence of a relation must be represented, in addition to the basic fact that the relation exists. For example, "A person has two hands" implies not only that a hand is part of every person, but also that in the case of "hands" there are exactly two such parts. This relation can be handled by using type-3 links, where the value of an attribute is a list of items, each of which is itself a property-list. The first item on such sub-property-lists is the flag PLIST, which indicates that a property-list follows. NAME is an attribute on each sub-property-list whose type-1 value is the principal object on the list. For example, after the system learns that "A person has two hands" and also "A finger is part of a person," the property-list of PERSON would contain the attribute-value pair:

(SUBPART, ((PLIST, NAME, HAND, NUMBER, 2) (PLIST, NAME, FINGER))).

In the interest of generality and uniformity type-3 links are the predominant mechanism for structuring the model.

Chapter IV: SIR Treatment of Restricted Natural Language

SIR must communicate with people: therefore the input and response languages of the SIR system should both be reasonably close to natural English. Since SIR utilizes a relational model, we are faced with the difficult problem of extracting relational information from natural language text.

I am primarily interested in the ability of a computer to store and utilize relational information in order to produce intelligent behavior. Although the linguistic problem of transforming natural language input into a usable form will have to be solved before we obtain a general semantic information retrieval system, it is independent of the representation and retrieval problems and therefore is considered beyond the scope of this paper.

In this chapter I shall describe briefly the background for the linguistic problem and the devices which SIR uses to bypass it, while still utilizing understandable English-like input and output.

A. Background

In the past ten to fifteen years much research has been done on the structure of natural languages, including English, for automatic processing by computer. In virtually every case, the form of the original text is restricted or pre-processed in some way to make it more amenable to automatic processing. Some of these studies were mentioned in Chapter II in connection with existing question-answering systems.

A recent paper by Bobrow (3) surveys various approaches and catalogues existing computer programs which automatically parse English text.

The object of most of these systems is to identify the classical grammatical structures of the sentences for purposes of linguistic analysis, mechanical translation, or information retrieval. Large dictionaries of parts of speech and grammatical rules are generally employed, and usually no consideration is given to the meanings (in any acceptable sense of the term "meaning") of the words and phrases involved.

A recent exception is the work at the National Bureau of Standards dealing with a "picture language machine" (10). Here the object is to determine whether a given English statement is a correct assertion about geometrical relationships in a given picture; therefore the "meaning" of the sentence is critical. The procedure used is to translate the English sentence into a logical statement involving geometric predicates, and then to test the truth of the logical statement by determining whether the relations specified by the predicates hold for the given picture.

In the SIR search and retrieval programs I am concerned with a problem similar to that of the picture language machine: namely, translating from English to a relational statement, and then determining how the relational statement affects the model. However, the SIR model is a data structure automatically built up on the basis of input relational statements, rather than an independently provided "picture." In the NBS system, the process of translating from English to the logical statement involves using a complete phrase-structure grammar for a fragment of English associated with picture descriptions. This seems like an extravagant approach, although it may turn out to be the one best capable of generalization. In the present version of SIR I am not concerned with constructing a formal logical statement of the relations recognized from the English sentence. Instead, the recognition programs directly invoke the appropriate storage or retrieval programs to deal with the relations recognized. I call the process of extracting relational information from English text "semantic parsing." The NBS work described above points to one rather expensive approach for obtaining this relational information. Charney (8) has studied the relation between sentence form and word meanings. Reichenbach (34) and Fries (16) also discuss the semantic parsing problem, and other approaches will undoubtedly be developed by linguists in the near future. It seems significant, although somewhat surprising, that the simple format-matching approach used in SIR, and discussed in part B below, is as effective as it is.

B. Input Sentence Recognition

SIR solves the semantic parsing problem by recognizing only a small number of sentence forms, each of which corresponds in specific ways to particular relations. The allowable input language is defined by a list of <u>rules</u>, each of which recognizes and operates upon a particular form of English sentence. Each sentence presented to SIR is tested by each rule in the list. The first rule applicable to the sentence determines the action taken by the system and immediately invokes a program to perform the action. If no rule is applicable, the sentence is ignored, except that the system makes an appropriate response (see Section C). A new rule may be added to the system, and thus the class of recognizable sentences may be enlarged, by executing the LISP function "addrule[x]" where x is the rule to be added. Let us consider the use of these rules in detail.

1) <u>Format matching procedure</u>:

The four components of a <u>rule</u> are a <u>format</u>, a list of the <u>variables</u> appearing in the format, a list of <u>applicability tests</u>, and an "<u>action</u>" list specifying the actions to be taken if the sentence satisfies all the tests. The format is simply a string of symbols which may be words. The list of variables contains those symbols which appear in the format which should be treated as variables. All other symbols in the format are constants. The first step in trying to apply a rule to a sentence is a "similarity test" between the sentence and the format of the rule to see whether the constants in the format all appear, in the same order, in the sentence. If they don't, the rule is rejected. If the sentence <u>is</u> similar to the format, the variables in the format are indentified with their corresponding substrings in the sentence.

The applicability tests are then applied, one to each substring matched by a variable. Each of these tests is the evaluation of a specified function of one argument, the corresponding substring. If the value of any of these function evaluations is the special LISP symbol "NIL" the substring is considered unsuitable and the entire rule is rejected. Otherwise, the system composes a list of the results of the applicability tests and communicates this list to the last part of the rule, the "action" list.

The first element of the action list is the name of a function which will act on the model to perform the operation required by the English sentence: create a link, test whether a particular relation holds by checking the existence of certain chains of links, or extract certain information from the model. The remaining elements of the action list are functions which, when applied to the list resulting from the applicability tests, produce arguments for the main action function.

For example, the semantic parsing of the sentence, "(A BOY IS A PERSON)" would be performed by a rule such as

((X IS A Y) (X Y) (ART ART) (SETR CAR CADR))

The format "(X IS A Y)" is indeed similar to the sentence "(A BOY IS A PERSON)" because the constants "IS" and "A" appear in both in the same order. Therefore the variable X is associated with the string "A BOY" and Y with "A PERSON." "ART" is the name of a function which tests whether its argument is a string of two symbols, the first of which is an indefinite article. If so, the value of "ART" is the second symbol in the string. Otherwise, the value of "ART" is "NIL."
In this case, the same applicability test function, "ART," is used for both matched substrings "A BOY" and "A PERSON." In both cases the

results of the test are positive, so the values of the two evaluations of "ART" are "BOY" and "PERSON," respectively. The system then composes the list of these values "(BOY, PERSON)", and proceeds to the "action" list. Here "SETR" is the SIR function which creates links indicating the existence of a set-inclusion relation between its two arguments. "CAR" and "CADR" are functions which obtain the arguments for "SETR" by extracting the first and second elements, respectively, from the value list "(BOY, PERSON)." After this final function "setr [BOY; PERSON]" is executed, the model will contain the relational information which the rule extracted from the sentence, "(A BOY IS A PERSON)." The recognition scheme does not distinguish between declarative sentences and questions; they each have their own formats and corresponding action fuctions. Of course, the effects of the action functions for questions are usually quite defferent from the effects of declarative-sentence functions. All action functions, as well as applicability tests, are programs which must be provided to the system along with each new rule.

Fig. 3 is a listing of all the rules included in the present version of SIR. The symbol"Q" is to be read as a question-mark. The significance of the "classify" function is explained in paragraph 2 below.

2) Ambiguities: The above translation from English sentence to action function can work only if a desired action is uniquely determined by each format. This is not really the case with many of the formats used for one of two reasons, which I call <u>format ambiguity</u>

and semantic ambiguity.

Format ambiguity is a programming device rather than a true ambiguity. It occurs when a single format (and rule) is used in order to save space and processing effort, even though several formats would be necessary to uniquely determine the required action. E.g., the sentence "Every boy is a person" specifies that the set "boy" is included in the set "person," while "The boy is a person" specifies that some particular element of the set "boy" is also an element of the set "person." These two types of sentences could be uniquely recognized by the formats, "Every x is a y" and "The x is a y." Instead, SIR uses a single format of the form, "z is a y." In the rule containing this format, the "action" function cannot be one which directly creates either a set-inclusion link, corresponding to the first of the above interpretations, or a set-membership link, corresponding to the second interpretation. Instead, the applicability test is the "classify" function which transmits to the action function an indicator of the nature of the article in the string matched by variable z, as well as the noun in the string. The action function then used is a "select" type of function which resolves the format ambiguity by examining the indicator supplied by "classify" and then invoking the correct action as a subroutine.

A more interesting case is that of semantic ambiguity, in which the ambiguity in desired action is due to the meanings of the words involved. Such an ambiguity cannot be resolved by using more-detailed formats. The example implemented in SIR involves the verb "to have,"

```
((X IS W)  (X  W) (CLASSIFY  CLASSIFY)  (SETR-SELECT  CAR  CADR))
((IS X Q)  (X)  (DECOMPOSE)  (SETRQ-SELECT  CAAR  CDAR))
((X OWNS Y) (X Y) (CLASSIFY  CLASSIFY)  (OWN-SELECT CADR  CAR))
((DOES X OWN Y Q) (X Y) (CLASSIFY CLASSIFY)  (OWN-SELECT CADR CAR))
((HOW MANY Y DOES X OWN Q) (Y X)(SING CLASSIFY) (OWN-SELECT CAR CADR))
((X IS Y PART OF Z)  (X Y Z)  (CLASSIFY A- CLASSIFY)
      (PARTR-SELECT  CAR  CADDR))
((X HAS AS A PART ONE Y)  (X Y)  (CLASSIFY  IDEN-1)
      (PARTRN-SELECT  CAR  CADR))
((THERE ARE Y ON X) (Y X) (NUM-Y CLASSIFY) (PARTRN-SELECT CADR CAR))
((THERE IS ONE Y ON X)  (Y  X)  (IDEN1  CLASSIFY)
      (PARTRN-SELECT  CADR  CAR))
((IS X PART OF Y Q)  (X Y)  ((LAMBDA  (J)  (CLASSIFY  (ALAST  J)))
      CLASSIFY)  (PARTRQ-SELECT  CAR  CADR))
((HOW MANY Y ARE TH ON X Q) (Y TH X) (SING THERE- CLASSIFY)
      (PARTRNQ-SELECT  CAR  CADDR))
((HOW MANY Y ARE PARTS OF X Q)  (Y X)  (SING  CLASSIFY)
      (PARTRNQ-SELECT  CAR  CADR))
((X HAS Y)  (X Y)  (CLASSIFY CLASSIFY)  (HAS-RESOLVE  CADR  CAR))
((X HAS W)  (X W)  (CLASSIFY  NUM-Y)  (HASN-RESOLVE  CADR  CAR))
((HOW MANY X DOES Y HAVE Q)  (X  Y)  (SING  CLASSIFY)
      (HAVE-RESOLVE  CAR  CADR)))
((X IS JUST TO THE RIGHT OF Y)  (X  Y)  (CLASSIFY  CLASSIFY)
      (JRIGHT-SELECT  CAR  CADR))
((X IS JUST TO THE LEFT OF Y)  (X  Y)  (CLASSIFY  CLASSIFY)
  (JRIGHT-SELECT  CADR  CAR)  )
((X IS TO THE RIGHT OF Y)  (X  Y) . (CLASSIFY  CLASSIFY)
  (RIGHT-SELECT  CAR  CADR)  )
((X IS TO THE LEFT OF Y)  (X  Y)  (CLASSIFY  CLASSIFY)
  (RIGHT-SELECT  CADR  CAR)  )
((IS X JUST TO THE RIGHT OF Y Q)  (X  Y)  (CLASSIFY  CLASSIFY)
  (JRIGHTQ-SELECT  CAR  CADR)  )
((IS X JUST TO THE LEFT OF Y Q)  (X  Y)  (CLASSIFY  CLASSIFY)
  (JRIGHTQ-SELECT  CADR  CAR)  )
((IS X TO THE RIGHT OF Y Q)  (X  Y)  (CLASSIFY  CLASSIFY)
  (RIGHTQ-SELECT CAR  CADR)  )
((IS X TO THE LEFT OF Y Q)  (X  Y)  (CLASSIFY  CLASSIFY)
  (RIGHTQ-SELECT  CADR  CAR)  )
((WHERE IS X Q)  (X)  (CLASSIFY)  (WHERE-SELECT  CAR))
((WHAT IS THE X OF Y Q)  (X Y)  (LOC CLASSIFY) (LOC-SELECT CADR))
```

FIGURE 3: SENTENCE RECOGNITION RULES

which may mean either "to have attached as parts" or "to own," e.g., "John has ten fingers" vs. "John has three marbles." In a case of semantic ambiguity the "action" function is a "resolve" type function which once again has the task of resolving the ambiguity and selecting the appropriate subroutine, rather than performing any action on the model directly. However, the ambiguity cannot be resolved on the basis of any information available in the original sentence. Instead, the ambiguity resolution depends upon word associations in the model which were created on the basis of previous, unambiguous sentences. Section V.B of this paper contains some examples and a discussion of the processes used, and further discussion of ambiguity can be found in section VII.D.

C. Output: Formation and Importance of Responses.

As with the input language, SIR avoids the problems of natural language processing in its responses. The response mechanism involves a set of built-in response formats. Although some generative grammar would probably be needed in a larger system, these response formats are adequate to demonstrate the use of the model and the ability of the present system to produce intelligible conversation.

Some of the responses are complete prepared statements, such as are frequently used as diagnostic comments in modern programming systems; e.g., the comment "The above statement is not recognized by the present system," which is printed if no rule is found to be applicable to the input sentence. Other responses must be completed by the programs which use them before being printed; e.g., the form, "I don't know whether ** is part of **," which is printed, after the **'s are appropriately replaced, in response to certain questions about part-whole relations.

One principle used in programming this system was that SIR should always make easily understandable reports of its actions. In particular, it should never fail to act on a new input sentence without presenting a reasonable explanation for its failure. Implementing this principle turned out to be easier than expected, for there always seemed to be only a small number of possible reasons for the failure of any one search procedure, and thus it was only necessary to provide a few response formats (and programs to use them). These responses, in turn, not only improved the conversational ability and thus the apparent intelligence of the system, but also greatly aided in debugging. SIR, in effect, frequently told me what it was doing wrong.

The conversation shown in Fig. 1 was produced by operating in an abbreviated-response mode in which SIR only prints directly relevant responses. The program can also operate in a mode in which SIR provides a running commentary of its activities, identifying functions used and commenting on every link created. Although less readable, this full-response mode was a significant program debugging aid. Fig. 4 shows the output for the dialogue of Fig. 1 in the alternate full-response mode.

```
(THE NEXT SENTENCE IS . .)
(EVERY BOY IS A PERSON)

(THE FUNCTION USED IS . .)
SETR-SELECT
((GENERIC . BOY) (GENERIC . PERSON))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETR
(BOY PERSON)
(ITS REPLY . .)
(I UNDERSTAND THE SUPERSET RELATION BETWEEN PERSON AND BOY)
(I UNDERSTAND THE SUBSET RELATION BETWEEN BOY AND PERSON)

(THE NEXT SENTENCE IS . .)
(A FINGER IS PART OF A HAND)

(THE FUNCTION USED IS . .)
PARTR-SELECT
((GENERIC . FINGER) (GENERIC . HAND))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
PARTR
(FINGER HAND)

(ITS REPLY . .)
(I UNDERSTAND THE SUBPART-OF-EACH RELATION BETWEEN FINGER AND HAND)
(I UNDERSTAND THE SUPERPART-OF-EACH RELATION BETWEEN HAND AND FINGER)

(THE NEXT SENTENCE IS . .)
(EACH PERSON HAS TWO HANDS)

(THE FUNCTION USED IS . .)
HASN-RESOLVE
((2 . HAND) (GENERIC . PERSON))
(THE REPLY . .)
(THE ABOVE SENTENCE IS AMBIGUOUS ** PLEASE RE-PHRASE IT)

(THE NEXT SENTENCE IS . .)
(THERE ARE TWO HANDS ON EACH PERSON)

(THE FUNCTION USED IS . .)
PARTRN-SELECT
((GENERIC . PERSON) (2 . HAND))
(THE REPLY . .)
(I UNDERSTAND THE SUPERPART-OF-EACH RELATION BETWEEN PERSON AND HAND)
(I REALIZE THE NUMBER RELATION BETWEEN 2 AND (PLIST NAME PERSON))
(I UNDERSTAND THE SUBPART-OF-EACH RELATION BETWEEN HAND AND PERSON)
(I REALIZE THE NUMBER RELATION BETWEEN [illegible])

(THE NEXT SENTENCE IS . .)
(HOW MANY FINGERS DOES JOHN HAVE Q)

(THE FUNCTION USED IS . .)
HAVE-RESOLVE
(FINGER (UNIQUE . JOHN))
(THE REPLY . .)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(I DON'T KNOW WHETHER FINGER IS PART OF JOHN)

(THE NEXT SENTENCE IS . .)
(JOHN IS A BOY)

(THE FUNCTION USED IS . .)
SETR-SELECT
((UNIQUE . JOHN) (GENERIC . BOY))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETRS
(JOHN BOY)
(ITS REPLY . .)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN JOHN AND BOY)
(I UNDERSTAND THE MEMBER RELATION BETWEEN BOY AND JOHN)

(THE NEXT SENTENCE IS . .)
(HOW MANY FINGERS DOES JOHN HAVE Q)

(THE FUNCTION USED IS . .)
HAVE-RESOLVE
(FINGER (UNIQUE . JOHN))

(THE REPLY . .)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(I KNOW THE SUPERPART-OF-EACH RELATION BETWEEN HAND AND FINGER)
(HOW MANY FINGERS PER HAND Q)

(THE NEXT SENTENCE IS . .)
(EVERY HAND HAS 5 FINGERS)

(THE FUNCTION USED IS . .)
HASN-RESOLVE
((5 . FINGER) (GENERIC . HAND))
(THE REPLY . .)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
```

FIGURE 4: SAMPLE CONVERSATION IN FULL-RESPONSE MODE

```
(I KNOW THE SUPERPART-OF-EACH RELATION BETWEEN HAND AND FINGER)
(I REALIZE THE NUMBER RELATION BETWEEN 5 AND (PLIST NAME HAND))
(I KNOW THE SUBPART-OF-EACH RELATION BETWEEN FINGER AND HAND)
(I REALIZE THE NUMBER RELATION BETWEEN 5 AND (PLIST NAME FINGER))


(THE NEXT SENTENCE IS . .)
(HOW MANY FINGERS DOES JOHN HAVE Q)

(THE FUNCTION USED IS . .)
HAVE-RESOLVE
(FINGER (UNIQUE . JOHN))
(THE REPLY . .)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(I KNOW THE SUPERPART-OF-EACH RELATION BETWEEN HAND AND FINGER)
(I KNOW THE SUPERPART-OF-EACH RELATION BETWEEN PERSON AND HAND)
(THE ANSWER IS 10)


(THE NEXT SENTENCE IS . .)
(HOW MANY AUTOMOBILES DOES JOHN HAVE Q)

(THE FUNCTION USED IS . .)
HAVE-RESOLVE
(AUTOMOBILE (UNIQUE . JOHN))
(THE REPLY . .)
(THE ABOVE SENTENCE IS AMBIGUOUS ** PLEASE RE-PHRASE IT)


(THE NEXT SENTENCE IS . .)
(WHO IS PRESIDENT OF THE UNITED STATES Q)

(STATEMENT FORM NOT RECOGNIZED)


(THE NEXT SENTENCE IS . .)
(THE BOY IS JUST TO THE LEFT OF THE TABLE)

(THE FUNCTION USED IS . .)
JRIGHT-SELECT
((SPECIFIC . TABLE) (SPECIFIC . BOY))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
JRIGHT
(TABLE BOY)
(ITS REPLY . .)
(G02840 IS A TABLE)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN G02840 AND TABLE)
(I UNDERSTAND THE MEMBER RELATION BETWEEN TABLE AND G02840)
(I REALIZE THE JRIGHT RELATION BETWEEN TABLE AND BOY)
(I REALIZE THE JLEFT RELATION BETWEEN BOY AND TABLE)


(THE NEXT SENTENCE IS . .)
(THE LAMP IS JUST TO THE LEFT OF THE TABLE)

(THE FUNCTION USED IS . .)
JRIGHT-SELECT
((SPECIFIC . TABLE) (SPECIFIC . LAMP))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
JRIGHT
(TABLE LAMP)
(ITS REPLY . .)
(G02841 IS A LAMP)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN G02841 AND LAMP)
(I UNDERSTAND THE MEMBER RELATION BETWEEN LAMP AND G02841)
(THE ABOVE STATEMENT IS IMPOSSIBLE)


(THE NEXT SENTENCE IS . .)
(THE TABLE IS TO THE RIGHT OF THE CHAIR)

(THE FUNCTION USED IS . .)
RIGHT-SELECT
((SPECIFIC . TABLE) (SPECIFIC . CHAIR))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
RIGHT
(TABLE CHAIR)
(ITS REPLY . .)
(G02842 IS A CHAIR)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN G02842 AND CHAIR)
(I UNDERSTAND THE MEMBER RELATION BETWEEN CHAIR AND G02842)
(I UNDERSTAND THE RIGHT RELATION BETWEEN TABLE AND CHAIR)
(I UNDERSTAND THE LEFT RELATION BETWEEN CHAIR AND TABLE)


(THE NEXT SENTENCE IS . .)
(WHAT IS THE RELATIVE POSITION OF A PERSON Q)

(THE FUNCTION USED IS . .)
LOC-SELECT
((GENERIC . PERSON))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
LOCATEG
(PERSON)
(ITS REPLY . .)
(THE LEFT-TO-RIGHT ORDER IS AS FOLLOWS)
(CHAIR (BOY TABLE))
```

FIGURE 4 (Cont.)

Chapter V: Behavior and Operation of SIR

In this chapter I shall give examples of typical conversations with SIR and explain the mechanisms which enable SIR to carry on its end of a conversation. These examples can frequently best be presented with the aid of logical notation, so formal symbols will be used when necessary. Explanations of the standard logical symbols are given in Appendix I.

Some knowledge of the LISP (21) programming language might be of aid in understanding the following pages. However, it should be sufficient for the reader to know the "fcn[a;b]" indicates that the function named "fcn" is to be applied to the symbols or symbolic expressions named "a" and "b" as arguments. This function of these arguments will have a value which is itself a symbolic expression, although the evaluation process may have side effects such as changing the model structure or printing comments. In more conventional programming terms, one may think of "fcn" as naming a subroutine, and "fcn[a;b]" representing the execution of the subroutine with "a" and "b" as input data. The creation of a single symbolic expression called the value is the principal result of the execution. This value of a function, which is a symbolic expression resulting from a computation, should not be confused with the value of an attribute, which is the entry following the attribute on a property-list.

A. Relations and Functions.

Each part of Fig. 5 is a conversation between a person and SIR, presented in the abbreviated-response mode described at the end of

Section IV.C. Each example illustrates the use of a different group of relations and their associated LISP functions in the SIR system. With minor exceptions the examples are cumulative, *i.e.*, later ones freely use functions introduced earlier but not conversely. These conversations are presented again as Appendix III in the full-response mode which identifies the functions used. In Fig. 5, the symbol "***." prefixes the input sentences; all other remarks are SIR responses.

The remainder of this section presents descriptions of all the significant functions mentioned in Appendix III in the order in which they are needed for the conversations. The functions are presented in groups which correspond to the various parts of Fig. 5, and which are identified by the principal attribute-links manipulated by the functions in the group.

Each function description consists of three parts: a *purpose*, a *method*, and a *procedure*. The *purpose* is a brief statement of the effect the function is designed to have. The *method* is an intuitive description of how the purpose should be achieved, and is usually presented in a mixture of English and logical notation for maximum clarity. Finally, the *procedure* is a description of how the method is implemented, and may be considered a rough flow-chart of the actual program. Notice that the relational structure of the model is the key internal feature of SIR which enables the *procedures* to implement the *methods* in a direct and efficient manner. These *methods*, in turn, determine the degree of SIR's ability to store facts and answer questions. Chapter VI will discuss how SIR's model, methods, and procedures could be generalized to produce a more powerful semantic information retrieval system.

a. SET-INCLUSION

```
(•••.  EVERY KEYPUNCH-OPERATOR IS A GIRL)
(I UNDERSTAND)

(•••.  ANY GIRL IS AN EXAMPLE OF A PERSON)
(I UNDERSTAND)

(•••.  IS A KEYPUNCH-OPERATOR A PERSON Q)
YES

(•••.  IS A PERSON A PERSON Q)
YES

(•••.  IS A PERSON A GIRL Q)
SOMETIMES

(•••.  IS A MONKEY A KEYPUNCH-OPERATOR Q)
(INSUFFICIENT INFORMATION)
```

b. SET-MEMBERSHIP

```
(•••.  MAX IS AN IBM-7094)
(I UNDERSTAND)

(•••.  AN IBM-7094 IS A COMPUTER)
(I UNDERSTAND)

(•••.  IS MAX A COMPUTER Q)
YES

(•••.  THE BOY IS AN MIT-STUDENT)
(G02840 IS A BOY)
(I UNDERSTAND)

(•••.  EVERY MIT-STUDENT IS A BRIGHT-PERSON)
(I UNDERSTAND)

(•••.  IS THE BOY A BRIGHT-PERSON Q)
YES

(•••.  JOHN IS A BOY)
(I UNDERSTAND)

(•••.  IS THE BOY A BRIGHT-PERSON Q)
(WHICH BOY . . (G02840 JOHN))
```

c. EQUIVALENCE

```
(•••.  THE MAN IS A JERK)
(G02840 IS A MAN)
(I UNDERSTAND)

(•••.  JACK IS A DOPE)
(I UNDERSTAND)

(•••.  JOHN IS JACK)
(I UNDERSTAND)

(•••.  IS JOHN A DOPE Q)
YES

(•••.  IS THE MAN A DOPE Q)
(INSUFFICIENT INFORMATION)

(•••.  JOHN IS THE MAN)
(I UNDERSTAND)

(•••.  IS THE MAN A DOPE Q)
YES

(•••.  JIM IS A MAN)
(I UNDERSTAND)

(•••.  IS THE MAN A DOPE Q)
(WHICH MAN . . (G02840 JIM))
```

FIGURE 5: SELECTED CONVERSATIONS

d. OWNERSHIP, GENERAL

```
(***.     EVERY FIREMAN OWNS A PAIR-OF-RED-SUSPENDERS)
(I UNDERSTAND)

(***.     DOES A PAIR-OF-RED-SUSPENDERS OWN A PAIR-OF-RED-SUSPENDERS Q)
(NO ** THEY ARE THE SAME)

(***.     DOES A DOCTOR OWN A PAIR-OF-RED-SUSPENDERS Q)
(INSUFFICIENT INFORMATION)

(***.     A FIRECHIEF IS A FIREMAN)
(I UNDERSTAND)

(***.     DOES A FIRECHIEF OWN A PAIR-OF-RED-SUSPENDERS Q)
YES
```

e. OWNERSHIP, SPECIFIC

```
(***.     ALFRED OWNS A LOG-LOG-DECITRIG)
(I UNDERSTAND)

(***.     A LOG-LOG-DECITRIG IS A SLIDE-RULE)
(I UNDERSTAND)

(***.     DOES ALFRED OWN A SLIDE-RULE Q)
YES

(***.     EVERY ENGINEERING-STUDENT OWNS A SLIDE-RULE)
(I UNDERSTAND)

(***.     VERNON IS A TECH-MAN)
(I UNDERSTAND)

(***.     A TECH-MAN IS AN ENGINEERING-STUDENT)
(I UNDERSTAND)

(***.     DOES VERNON OWN A SLIDE-RULE Q)
YES

(***.     DOES AN ENGINEERING-STUDENT OWN THE LOG-LOG-DECITRIG Q)
(G02840 IS A LOG-LOG-DECITRIG)
(INSUFFICIENT INFORMATION)

(***.     ALFRED IS A TECH-MAN)
(I UNDERSTAND)

(***.     DOES AN ENGINEERING-STUDENT OWN THE LOG-LOG-DECITRIG Q)
YES
```

FIGURE 5 (Cont.)

f. PART-WHOLE, GENERAL

```
(***.     A NOSE IS PART OF A PERSON)
(I UNDERSTAND)

(***.     A NOSTRIL IS A PART OF A NOSE)
(I UNDERSTAND)

(***.     A PROFESSOR IS A TEACHER)
(I UNDERSTAND)

(***.     A TEACHER IS A PERSON)
(I UNDERSTAND)

(***.     IS A NOSTRIL PART OF A PROFESSOR Q)
YES

(***.     IS A NOSE PART OF A NOSE Q)
(NO , PART MEANS PROPER SUBPART)

(***.     A PERSON IS A LIVING-CREATURE)
(I UNDERSTAND)

(***.     IS A NOSTRIL PART OF A LIVING-CREATURE Q)
SOMETIMES

(***.     IS A LIVING-CREATURE PART OF A NOSE Q)
(NO , NOSE IS SOMETIMES PART OF LIVING-CREATURE)
```

g. PART-WHOLE, SPECIFIC

```
(***.     A VAN-DYKE IS PART OF FERREN)
(I UNDERSTAND)

(***.     A VAN-DYKE IS A BEARD)
(I UNDERSTAND)

(***.     IS A BEARD PART OF FERREN Q)
YES

(***.     A CRT IS A DISPLAY-DEVICE)
(I UNDERSTAND)

(***.     A CRT IS PART OF THE PDP-1)
(G02840 IS A PDP-1)
(I UNDERSTAND)

(***.     SAM IS THE PDP-1)
(I UNDERSTAND)

(***.     A SCREEN IS PART OF EVERY DISPLAY-DEVICE)
(I UNDERSTAND)

(***.     IS A SCREEN PART OF SAM Q)
YES

(***.     A BEARD IS PART OF A BEATNIK)
(I UNDERSTAND)

(***.     EVERY COFFEE-HOUSE-CUSTOMER IS A BEATNIK)
(I UNDERSTAND)

(***.     BUZZ IS A COFFEE-HOUSE-CUSTOMER)
(I UNDERSTAND)

(***.     IS A BEARD PART OF BUZZ Q)
YES
```

FIGURE 5 (Cont.)

h. NUMBER

```
(***.      A BOY IS A PERSON)

(I UNDERSTAND)

(***.      JOHN IS A BOY)

(I UNDERSTAND)

(***.      A FINGER IS PART OF A HAND)

(I UNDERSTAND)

(***.      HOW MANY FINGERS DOES JOHN HAVE Q)

(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(I DON*T KNOW WHETHER FINGER IS PART OF JOHN)

(***.      THERE IS ONE HAND ON EACH ARM)

(I UNDERSTAND)

(***.      THERE ARE TWO ARMS ON A PERSON)

(I UNDERSTAND)

(***.      HOW MANY FINGERS DOES JOHN HAVE Q)

(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
((HOW MANY FINGER PER HAND Q))

(***.      A HAND HAS 5 FINGERS)

(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(I UNDERSTAND)

(***.      HOW MANY FINGERS DOES JOHN HAVE Q)

(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(THE ANSWER IS 10)
```

FIGURE 5 (Cont.)

i. LEFT-TO-RIGHT POSITION

```
(***.      THE TELEPHONE IS JUST TU THE RIGHT OF THE BOOK)

(G02840 IS A TELEPHONE)
(G02841 IS A BOOK)
(I UNDERSTAND)

(***.      THE TELEPHONE IS JUST TO THE LEFT OF THE PAD)

(G02842 IS A PAD)
(I UNDERSTAND)

(***.      IS THE PAD JUST TO THE RIGHT OF THE BOOK Q)

NO

(***.      IS THE BOOK TO THE LEFT OF THE PAD Q)

YES

(***.      THE PAD IS TO THE RIGHT OF THE TELEPHONE)

(THE ABOVE STATEMENT IS ALREADY KNOWN)

(***.      THE PAD IS TO THE LEFT OF THE TELEPHONE)
(THE ABOVE STATEMENT IS IMPOSSIBLE)

(***.      THE ASH-TRAY IS TO THE LEFT OF THE BOOK)

(G02843 IS A ASH-TRAY)
(I UNDERSTAND)

(***.      THE PENCIL IS TO THE LEFT OF THE PAD)

(G02844 IS A PENCIL)
(I UNDERSTAND)

(***.      THE PAPER IS TO THE RIGHT OF THE TELEPHONE)

(G02845 IS A PAPER)
(I UNDERSTAND)

(***.      WHERE IS THE PAD Q)

(JUST TO THE RIGHT OF THE TELEPHONE)
(SOMEWHERE TO THE RIGHT OF THE FOLLOWING . . (PENCIL))

(***.      WHAT IS THE POSITION OF THE PAD Q)

(THE LEFT-TO-RIGHT ORDER IS AS FOLLOWS)
(ASH-TRAY (BOOK TELEPHONE PAD) PAPER)
(TO FURTHER SPECIFY THE POSITIONS YOU MUST INDICATE WHERE THE PENCIL IS WITH RESPECT TO THE ASH-TRAY)

(***.      THE BOOK IS JUST TO THE RIGHT OF THE ASH-TRAY)

(I UNDERSTAND)

(***.      WHAT IS THE POSITION OF THE PAD Q)

(THE LEFT-TO-RIGHT ORDER IS AS FOLLOWS)
(PENCIL (ASH-TRAY BOOK TELEPHONE PAD) PAPER)

(***.      A TELEPHONE IS AN AUDIO-TRANSDUCER)

(I UNDERSTAND)

(***.      A DIAPHRAGM IS PART OF AN AUDIO-TRANSDUCER)

(I UNDERSTAND)

(***.      WHERE IS A DIAPHRAGM Q)

(JUST TO THE LEFT OF THE PAD)
(JUST TO THE RIGHT OF THE BOOK)
(SOMEWHERE TO THE LEFT OF THE FOLLOWING . . (PAPER))
```

FIGURE 5 (Cont.)

Operation of functions:

a) Attributes: SUBSET, SUPERSET

1. setr[x;y]

purpose: To specify in the model that set x is included in set y.

method: Create a type-3 link between x and y which indicates set-inclusion.

procedure:
a. Add "(PLIST NAME x)" to the value list of attribute "SUBSET" of y.
b. Add "(PLIST NAME y)" to the value list of attribute "SUPERSET" of x.
c. Respond "(I UNDERSTAND)"

2. setrq[x;y]

purpose: To reply as to whether an arbitrary element of set x is an element of set y.

method: A member of x is considered to be a member of y if the sets x and y are identical; or if there is a chain of explicit set-inclusion links proving that x is a subset of y, i.e., if there exists a (possibly empty) sequence of sets v,w, ... z such that

$$x \subset v \wedge v \subset w \wedge \ldots \wedge z \subset y.$$

A member of x is "sometimes" in y if there is a chain of explicit set-inclusion links proving that y is a subset of x.

procedure:
a. If x=y, respond "YES".
b. If there is a path from x to y through type-3 links following the attribute "SUPERSET", respond "YES".
c. If there is a path from y to x through type-3 links following the attribute "SUPERSET", respond "SOMETIMES".
d. Otherwise, respond "(INSUFFICIENT INFORMATION)".

b) Attributes: MEMBER, ELEMENTS

1. setrs[x;y]

purpose: To specify in the model that x is a member of the set y.

method: Create a type-3 link between x and y which indicates set-membership.

procedure:

a. Add "(PLIST NAME y)" to the value list of attribute "MEMBER" of x.

b. Add "(PLIST NAME x)" to the value list of attribute "ELEMENTS" of y.

c. Respond "(I UNDERSTAND)".

2. setrsq[x;y]

purpose: To reply as to whether x is a member of the set y.

method: Reply "YES" if the folling is true:

$(\exists u)[[u{=}x \lor [u \text{ is equivalent}^{*} \text{ to } x]] \land$
$[[\text{there is a link indicating that } u \text{ is a member of } y] \lor$
$[(\exists z)[[\text{there is a link indicating that } u \text{ is a member of } z] \land$
$[\text{any member of set } z \text{ is a member of set } y]]]]]$

procedure:

a. Make a list of the items connected to x by a type-3 link following the attribute "MEMBER".

b. If y is on the list, respond "YES".

c. If, for any member z of the list, setrsq[z;y]=YES, respond "YES".

d. Repeat steps (a) through (c) with x replaced by each item equivalent* to x (if any) until a "YES" response is made.

e. Otherwise respond "(INSUFFICIENT INFORMATION)".

3. setrs1[x;y]

purpose: To specify in the model that the unique element (if any) of the set x is also an element of the set y.

method: Create a type-3 link from the unique element of x to y which indicates set-membership. If x has more than one element, do not set up any link.

procedure:

a. Compute u = specify[x].

b. If u = NIL, terminate.

c. Otherwise execute setrs[u;y].

4. specify[x]

purpose: To determine the unique element, if any, of the set x.

method: If x has one element, find its name. If x has no elements, create one and give it a name. If x has more than one element, ask which one and indicate failure.

*See part (c) for an explanation of "equivalent".

procedure:

a. Get the value list of the attribute "ELEMENTS" of x.

b. If there is no list, create a new symbol u, respond "(u IS A x)", execute setrs[u;x], and return u as the value of specify[x].

c. If there is just one element named on the list, or if all the elements are equivalent, return the name of the first element as the value of specify[x].

d. Otherwise respond "(WHICH x ... v)", where v is a list of names of the elements, and return "NIL" as the value of specify[x].

5. setrslq[x;y]

purpose: To reply as to whether the unique element, if any, of the set x, is a member of the set y.

method: Determine the element referred to and apply setrsq.

procedure:

a. Compute u = specify[x].

b. If u = NIL, terminate.

c. Execute setrsq[u;y].

c) Attribute: EQUIV

1. equiv[x;y]

purpose: To specify in the model that x and y are equivalent.

method: Create a type-2 link between x and y which indicates equivalence.

procedure:

a. Add x to the value list of attribute "EQUIV" of y.

b. Add y to the value list of attribute "EQUIV" of x.

c. Respond "(I UNDERSTAND)".

2. equivl[x;y]

purpose: To specify in the model that x is equivalent to the unique element of the set y.

method: Determine the element referred to and apply equiv.

procedure:

a. Compute u = specify[y].

b. If u = NIL, terminate.

c. Execute equiv[x;u].

d) Attributes: OWNED-BY-EACH, POSSESS-BY-EACH

1. ownr[x;y]

purpose: To specify in the model that every member of set y owns some member of set x.

method: Create a type-3 link between x and y which indicates the ownership relation between their members.

procedure:
a. Add "(PLIST NAME y)" to the value list of attribute "OWNED-BY-EACH" of x.
b. Add "(PLIST NAME x)" to the value list of attribute "POSSESS-BY-EACH" of y.
c. Respond "(I UNDERSTAND)".

2. ownrq[x;y]

purpose: To reply as to whether an arbitrary member of set y owns some member of set x.

method: The answer is "YES" if $x \neq y$, and
$(\exists z)[y=z \vee [y \text{ is a subset of } z]] \wedge$
[there exists the appropriate ownership link between x and z]]

procedure:
a. If x=y, respond "(NO ** THEY ARE THE SAME)".
b. Create the list ℓ containing y and all sets u for which there is a path from y to u through type-3 links following the attribute "SUPERSET".
c. If any element of ℓ contains a type-3 link to x following the attribute "POSSESS-BY-EACH", respond "YES".
d. Otherwise respond "(INSUFFICIENT INFORMATION)".

e) Attributes: OWNED, POSSESS

1. ownrgu[x;y]

purpose: To specify in the model that y owns a member of the set x.

method: Create a type-3 link between x and y which indicates the intended ownership relation.

procedure:
a. Add "(PLIST NAME x)" to the value list of attribute "POSSESS" of y.
b. Add "(PLIST NAME y)" to the value list of attribute "OWNED" of x.
c. Respond "(I UNDERSTAND)".

2. ownrguq[x;y]

purpose: To reply as to whether y owns a member of set x.

method: The reply is "YES" if there is a link indicating that y owns a member of x or of some subset of x;or if

$(\exists z)[[y$ is a member of $z] \wedge (\exists u)[[u=z \vee z \subset u] \wedge$
[there is a link indicating that every member of set u owns a member of set x]]]

procedure:

a. If there is a link indicating an x is owned by y, respond "YES".

b. Consider each set z for which there is a link indicating that y owns a member of z. If, for any z, setrq[z;x]=YES, respond "YES".

c. Consider each set z such that there is a link indicating y is an element of z.

d. For each z, construct a list ℓ containing every set u for which setrq[z;u]=YES.

e. Compute m = the list of all sets v such that there is a type-3 link from x to v following the attribute "OWNED-BY-EACH".

f. If, for some z, the intersection of ℓ and m is non-empty, respond "YES".

g. Otherwise, respond "(INSUFFICIENT INFORMATION)".

3. ownrsgq[x;y]

purpose: To reply as to whether the unique element of the set x is owned by some element of the set y.

method: Determine that a unique element of x exists. Then, the reply is "YES" if

$(\exists z)[[$there is a link indicating that a member of set x is owned by $z] \wedge$
$(\exists v)[[v=z \vee [v$ is equivalent to $z]] \wedge$
$(\exists w)[[$there is a link indicating that y is an element of $w] \wedge$
[there are links indicating that w is a subset of y]]]]

procedure:

a. Compute u = specify[x]

b. If u = NIL, terminate.

c. Generate the individuals w which are linked to x as type-3 values of the attribute "OWNED".

d. For each w, generate the sets z which w, and any individual equivalent to w, is a member of.

e. If, for some z, setrq[z;y] = YES, respond "YES".

f. Otherwise respond "(INSUFFICIENT INFORMATION)".

f) Attributes: SUPERPART-OF-EACH, SUBPART-OF-EACH

1. partr[x;y].

purpose: To specify in the model that every element of set x is part of some element of set y.

method: Create a type-3 link between x and y which indicates the part-whole relation between their members.

procedure:

a. Add "(PLIST NAME y)" to the value list of attribute "SUPERPART-OF-EACH" of x.

b. Add "(PLIST NAME x)" to the value list of attribute "SUBPART-OF-EACH)" of y.

c. Respond "(I UNDERSTAND)".

2. partrq[x;y].

purpose: To reply as to whether an arbitrary member of set x is a part of some member of set y.

method: No element may be part of itself. Reply "YES" if

(∃w)[[there is a chain of links indicating that an arbitrary member of set x is part of some member of w] ∧ [[y=w ∨ [[there is a chain of links indicating that y is a subset of w]]].

Reply "SOMETIMES" if

(∃w)[[there is a chain of links indicating that an arbitrary member of set x is part of some member of w] ∧ [there is a chain of links indicating that w is a subset of y]].

Reply "NO" if an arbitrary member of set y is always or sometimes a part of some member of set x.

procedure:

a. If x=y, respond "(NO, THEY ARE THE SAME)".

b. Generate those sets w which can be reached from x through a chain of type-3 links following the attribute "SUPERPART-OF-EACH".

c. If, for some w, setrq[y;w] = YES or SOMETIMES, respond "YES" or "SOMETIMES", respectively.

d. If the response for partrq[y;x] would be YES or SOMETIMES, respond "(NO, y IS PART OF x)" or "(NO, y IS SOMETIMES PART OF x)", respectively.

e. Otherwise respond "(INSUFFICIENT INFORMATION)".

g) Attributes: SUBPART, SUPERPART

1. partrgu[x;y]

purpose: To specify in the model that some element of set x is a part of the individual y.

method: Create a type-3 link between x and y which indicates the appropriate part-whole relation.

procedure:
a. Add "(PLIST NAME x)" to the value list of attribute "SUBPART" of y.
b. Add "(PLIST NAME y)" to the value list of attribute "SUPERPART" of x.
c. Respond "(I UNDERSTAND)".

2. partrgs[x;y]

purpose: To specify in the model that some element of set x is a part of the unique element, if any, of the set y.

method: Determine z, the unique element of y. Then specify that some element of x is part of z.

procedure:
a. Compute z = specify[y].
b. If z = NIL, terminate.
c. Else, compute partrgu[x;z].

3. partrguq[x;y]

purpose: To reply as to whether some element of set x is part of the individual y.

method: A member of x is a part of y if

(∃u)[[u=y ∨ [u is equivalent to y]] ∧
[(∃w)[[there is a link indicating that an element of w is a subpart of u] ∧
[[w=x ∨ [there are links indicating that w is a subset of x] ∨
(∃z)[[there are links indicating that every element of z has some element of x as a part] ∧
[w=z ∨ [there are links indicating that w is a subset of z]]]] ∨
[(∃z)[[u is an element of set z] ∧
[(∃v)[there are links indicating that z is a subset of v]]]]]]]]]

procedure:
a. Generate those nodes w which can be reached from y, or from any node equivalent to y, by a chain of type-3 links following the attribute "SUBPART."

b. If, for any w, setrq[w;x]=YES, respond "YES".

c. Otherwise, generate those nodes z which can be reached from x by a chain of type-3 links following the attribute "SUPERPART-OF-EACH".

d. If, for any z and any w, setrq[w;z]=YES respond "YES".

e. Otherwise, compute the list ℓ of sets for which there is a type-3 link from y, or any node equivalent to y, following the attribute "MEMBER".

f. Generate the nodes v which can be reached by a chain of type-3 links from x following the attribute, "SUPERPART-OF-EACH".

g. If, for any v and any u in ℓ, setrq[u;v]=YES, respond "YES".

h. Otherwise, respond "(INSUFFICIENT INFORMATION)".

4. partrss[x;y]

purpose: To specify in the model that the unique element, if any, of set x is part of the unique element, if any, of set y.

method: Identify the unique elements u and v of sets x and y, respectively. Specify that some element of set x is part of the individual v. Then create a type-2 link from the appropriate type-3 link from x to u, specifying which element of x is involved.

procedure:

a. Compute v=specify[b], and u=specify[a].

b. If u or v = NIL, terminate.

c. Execute partrgu[x;v].

d. Add u to the value list of attribute "ELEMENTS" on that member of the "SUPERPART" value list of x which refers to y.

e. Respond "(I UNDERSTAND)".

5. partrsgq[x;y]

purpose: To reply as to whether the unique element of set x is part of some element of set y.

method: The answer is "YES" if there exists a unique element z of set x and if

(∃w)[[there is a link indicating that some x is part of w]∧
(∃u)[[u=w∨ u is equivalent to w]∧
(∃v)[[there is a link indicating that u is an element of v]∧
[[y=v]∨[there are links indicating that y is a subset of v]∨
(∃q)[[there are links indicating that every y
is part of some q]∧[[v=q]∨
[there are links indicating that v is a subset of q]]]]]]]

procedure:

a. Compute z = specify[x].

b. If z = NIL, terminate.

c. Generate those nodes w which can be reached from x by a type-3 link following the attribute "SUPERPART".

d. For each w compute the list ℓ of those sets which w, or any set equivalent to w, is a member of.

e. If y is in ℓ, respond "YES".

f. If, for any v∈ℓ, setrq[y;v] = YES, respond "YES".

g. Otherwise, generate those nodes q which can be reached from y by a type-3 link following the attribute "SUPERPART-OF-EACH".

h. If, for any q, setrq[v;q] = YES, respond "YES".

i. Otherwise respond "(INSUFFICIENT INFORMATION)".

h) Attribute: NUMBER

1. partrn[x;y;n]

purpose: To specify in the model that there are n elements of the set x which are parts of every element of set y.

method: Create a type-3 link between x and y specifying that an element of x is part of some element of y. Create type-1 links associating the number n with that type-3 link.

procedure:

a. Execute partr[x;y].

b. Add "(NUMBER n)" to both the list which was added to the value list of attribute "SUBPART-OF-EACH" of y, and the list which was added to the value list of attribute "SUPERPART-OF-EACH" of x.

2. partrnu[x;y;n]

purpose: To specify in the model that there are n elements of set x which are parts of individual y.

method: Create a type-3 link between x and y which indicates that some element of set x is part of y. Create type-1 links associating the number n with that type-3 link.

procedure:

a. Execute partrgu[x;y].

b. Add "(NUMBER n)" to both the list which was added to the the value list of attribute "SUBPART" of y, and the list which was added to the value list of attribute "SUPERPART" of x.

3. partrnuq[x;y]

purpose: to reply as to how many elements of the set x are parts of the individual y.

method: If

$(\exists u)$[[there is a link indicating that an element of u is part of y]$\wedge$

[[u=x] $\vee$ $(\exists v)$[[there is a chain of links indicating that

a v is part of every u]$\wedge$ [[x=v]$\vee$

[there is a chain of links indicating x is a subset of v]]]]$\vee$

$(\exists u)$[[there is a link indicating that y is an element of set u]

$(\exists v)$[[there is a chain of links indicating that a v is a

part of every u]$\wedge$ [[x=v]$\vee$

[there is a chain of links indicating that x is a

subset of v]]]],

then the answer is the product of the values of the type-1 links following the attribute "NUMBER", associated with each type-3 link used in proving the required part relation. If any such "NUMBER" attribute is missing, the reply should explicitly request it. If the part-whole relation cannot be established, the reply indicates that fact.

procedure:

a. Follow the procedure of partrguq[x;y] until links are found which warrent a "YES" response. Save a list ℓ of all required links which follow the attribute "SUBPART" or "SUPERPART-OF-EACH".

b. If no such list can be found, respond "(I DON*T KNOW WHETHER x IS PART OF y)".

c. For each element α of ℓ, where α specifies a "SUPERPART-OF-EACH" link from u to v, get the value of the attribute "NUMBER" of α. If, for some α, no such value exists, respond "(HOW MANY (u PER v))".

d. Compute n = the product of the numbers obtained above. Respond "(THE ANSWER IS n)".

i) Attributes: LEFT, RIGHT, JLEFT, JRIGHT

1. jright[x;y]

purpose: To specify in the model that the unique element of set x is located just to the right of the unique element of set y.

method: Check whether the statement is consistent with existing knowledge; i.e., that nothing is known to be between x and y and that y is not known to be to the right of x. If it is not consistent, complain. Otherwise, create a type-1 link indicating the positional relation.

procedure:

a. If specify[x] or specify[y] = NIL, terminate.

b. If there is already a type-1 link from y to x following the attribute "JRIGHT", respond "(THE ABOVE STATEMENT IS ALREADY KNOWN)".

c. If it can be proven that y is to the right of x, i.e., if rightp[y;x]=T ; or if there is any type-1 link from y following the attribute "JRIGHT"; or if there is any type-1 link from x following the attribute "JLEFT"; then respond "(THE ABOVE STATMENT IS IMPOSSIBLE)".

d. If rightp[x;y]=T, and there does not exist a direct type-2 link from y to x following the attribute "RIGHT", respond "(THE ABOVE STATEMENT IS IMPOSSIBLE)".

e. Otherwise, create a type-1 link from y to x following the attribute "JRIGHT"; create a type-1 link from x to y following the attribute "JLEFT"; and respond "(I UNDERSTAND)".

2. rightp[x;y]

purpose: To test whether it is known that the x is located to the right of the y.

method: "rightp[x;y]" is defined recursively as follows: If there is no type-1 link from y following the attribute "JRIGHT", and no type-2 link from y following the attribute "RIGHT", the value of "rightp[x;y]" is NIL; if either of the above links exists and links to x, the value is T. Otherwise the value is the disjunction of the values of "rightp[x;y]" for all u which are linked to y by one of the above links.

procedure:

a. Compute u, the value of the type-1 link from y following the attribute "JRIGHT".

b. If u=x, value is T; if there is no u, go to step d.

c. If rightp[x;u] = T, the value is T.

d. Compute ℓ, the value of the type-2 link from y following the attribute "RIGHT".

e. If x is a member of list ℓ, the value is T; if there is no ℓ, the value is NIL.

f. If, for any $v \in \ell$, rightp[x;v]=T, the value is T; otherwise the value is NIL.

note: "T" and "NIL" are special LISP symbols standing for "true" and "false," respectfully.

3. right[x;y]

purpose: To specify in the model that the unique element of set x is located to the right of the unique element of set y.

method: Check whether the statement is consistent with existing knowledge. If so, create a type-2 link indicating the positional relation. Otherwise, complain.

procedure:

a. If specify[x]=NIL or specify[y]=NIL, terminate.

b. If rightp[x;y]=T, respond "(THE ABOVE STATEMENT IS ALREADY KNOWN)."

c. If rightp[y;x]=T, respond "(THE ABOVE STATEMENT IS IMPOSSIBLE)".

d. Otherwise, create a type-2 link from y to x following the attribute "RIGHT"; create a type-2 link from x to y following the attribute "LEFT"; and respond "(I UNDERSTAND)".

4. jrightsaq[x;y]

purpose: To reply as to whether the x is located just to the right of the y.

method: Determine whether the links in the model indicate that x is just to the right of y, x cannot be just to the right of y, or neither.

procedure:

a. If specify[x]=NIL or specify[y]=NIL, terminate.

b. If there is a type-1 link from y to x following the attribute "JRIGHT", respond "YES".

c. If rightp[y;x]=T; or if there is any type-1 link from y following the attribute "JRIGHT"; or if there is any type-1 link from x following the attribute "JLEFT"; then respond "NO ".

d. If rightp[x;y]=T and there does not exist a direct type-2 link from y to x following the attribute "RIGHT", respond "NO".

e. Otherwise, respond "(INSUFFICIENT INFORMATION)".

5. rightssq[x;y]

purpose: To reply as to whether the x is located to the right of the y.

method: Determine whether the links in the model indicate that x is to the right of y, to the left of y, or neither.

procedure:

a. If specify[x]=NIL or specify[y]=NIL, terminate.

b. If rightp[x;y]=T, respond "YES".

c. If rightp[y;x]=T, respond "NO".

d. Otherwise, respond "(INSUFFICIENT INFORMATION)".

6. wheres[x]

purpose: To determine the locations of those objects which have been positioned with respect to the unique element of the set x.

method: Reply with the information provided by each positional link associated with x.

procedure:

a. If specify[x]=NIL, terminate.

b. Compute u = the value of the type-1 link from x following the attribute "JLEFT"; v = the value of the type-1 link from x following the attribute "JRIGHT"; ℓ = the value of the type-2 link from x following the attribute "LEFT"; and m = the value of the type-2 link from x following the attribute "RIGHT".

c. If u, v, ℓ, and m all do not exist, respond "(NO POSITION IS KNOWN)".

d. If u does not exist, go to step f.

e. Respond, "(JUST TO THE RIGHT OF THE u)", and go to the next step.

f. If v does not exist, go to step h.

g. Respond, "(JUST TO THE LEFT OF THE v)", and go to the next step.

h. If ℓ does not exist, go to step j.

i. Respond, "(SOMEWHERE TO THE RIGHT OF THE FOLLOWING . . . ℓ)", and go to the next step.

j. If m does not exist, terminate.

k. Respond, "(SOMEWHERE TO THE LEFT OF THE FOLLOWING . . m)".

7. locates[x]

purpose: To determine the location of the unique element of set x with respect to as many other objects as possible.

method: Construct a diagram of the left-to-right order of objects by searching through all chains of positional links starting from x and proceeding recursively. The form of the diagram is a list, with objects known to be adjacent appearing in sublists. If no positional links from x exist or if a well-ordering cannot be determined, make an appropriate comment.

procedure:

a. If specify[x]=NIL, terminate.

b. Set the initial diagram g="(x)".

c. Compute u = the value of the type-1 link from x following the attribute "JRIGHT". If no u exists or if u is already in g, go to step f.

d. Insert u just to the right of x in g, i.e., insert u right after x in a sublist of g.

e. Replace g by the result of executing this procedure starting from step c, with the current value of u replacing the argument x and the current value of g as the diagram.

f. Repeat step c, for the attribute "JLEFT". In case of failure, go to step i.

g. Insert u just to the left of x in g.

h. Repeat step e.

i. Compute ℓ = the value of the type-2 link from x following the attribute "RIGHT". If no ℓ exists, go to step ℓ.

j. For each $m \in \ell$: If m is already in the current g, ignore it; if there exists a v in g which is the object (or first object on a sublist) following x (or a sublist containing x), go to step k. Otherwise insert m after x (or the sublist containing x) in g, and repeat step e, with the current value of m replacing x. When all $m \in \ell$ have been treated go to step ℓ.

k. If rightp[v;m]=T, insert m after x and continue with the next m in step j. If rightp[m;v]=T, then just for this value of m replace x by v and continue as in step j. Otherwise, respond

"(THE LEFT-TO-RIGHT ORDER IS)
g
(TO FURTHER SPECIFY THE POSITIONS YOU MUST INDICATE WHERE THE m IS
WITH RESPECT TO THE v)".

l. Perform operations analogous to i, j, and k for the attribute "LEFT" of x.

m. If the current g="(x)", respond "(NO RELATIVE POSITION IN KNOWN)".

n. Otherwise respond, "(THE LEFT-TO-RIGHT ORDER IS) g".

8. whereg[x]

purpose: To determine the locations of those objects which have been positioned with respect to some element of set x.

method: Find an object u of which an x is an example or a part, and which has positional links. Then find the locations of those objects which have been positioned with respect to u.

procedure:

a. If x has any positional links, i.e., if the attributes "JRIGHT", "JLEFT", "RIGHT", and "LEFT" of x are not all missing, execute wheres[x].

b. If
(∃u)[[there is a sequence of links following the attribute "SUPERPART-OF-EACH" from x to u] ∧ [u has at least one positional link]],
then execute wheres[u].

c. If the hypotheses of step b. hold for the attribute "SUBSET", execute wheres[u].

d. If
(∃u)[[there is a sequence of links following the attribute "SUPERPART-OF-EACH" from x to u] ∧ (∃w)[[there is a sequence of links following the attribute "SUBSET" from u to w] ∧ [w has at least one positional link]]],
then execute wheres[w].

e. Otherwise respond "(NO RELATIVE POSITION IS KNOWN)".

B. Special Features.

This section discusses the sample conversations for Fig. 6 which illustrate three special features of the SIR system. The first two -- the exception principle and resolving ambiguities -- illustrate how SIR can be used to simulate various aspects of human linguistic behavior. The third feature -- streamlining -- demonstrates one way in which SIR can automatically modify its model structure in order to save computer memory space.

a) Exception principle: General information about "all the elements" of a set is considered to apply to particular elements only in the absence of more specific information about those elements. Thus it is not necessarily contradictory to learn that "mammals are land animals" and yet "a whale is a mammal which always lives in water." In the program, this idea is implemented by always referring for desired information to the property-list of the individual concerned before looking at the descriptions of sets to which the individual belongs.

The justification for this departure from the no-exception principles of Aristotelian logic is that this precedence of specific facts over background knowledge seems to be the way people operate, and I wish the computer to communicate with people as naturally as possible.

The present program does not experience the uncomfortable feeling people frequently get when they must face facts like "a whale is a mammal which lives in water although mammals as a rule live on land." However, minor programming additions to the present system could

a. EXCEPTION PRINCIPLE

```
(***.     THERE ARE 5 FINGERS ON EVERY HAND)
(I UNDERSTAND)

(***.     THERE ARE TWO HANDS ON A PERSON)
(I UNDERSTAND)

(***.     A BOY IS A PERSON)
(I UNDERSTAND)

(***.     TOM IS A BOY)
(I UNDERSTAND)

(***.     DICK IS A BOY)
(I UNDERSTAND)

(***.     HARRY IS A BOY)
(I UNDERSTAND)

(***.     TOM HAS NINE FINGERS)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(I UNDERSTAND)

(***.     DICK HAS ONE HAND)

(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(I UNDERSTAND)

(***.     HOW MANY FINGERS DOES TOM HAVE Q)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(THE ANSWER IS 9)

(***.     HOW MANY FINGERS DOES DICK HAVE Q)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(THE ANSWER IS 5)

(***.     HOW MANY FINGERS DOES HARRY HAVE Q)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(THE ANSWER IS 10)

(***.     HOW MANY FINGERS DOES JOE HAVE Q)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(I DON*T KNOW WHETHER FINGER IS PART OF JOE)
```

FIGURE 6: SPECIAL FEATURES

b. RESOLVING AMBIGUITIES

```
(***.     JOHN IS A PERSON)

(I UNDERSTAND)

(***.     DICK IS A PERSON)

(I UNDERSTAND)

(***.     A CHAIN IS PART OF A BICYCLE)

(I UNDERSTAND)

(***.     THE POWER-SAW HAS A CHAIN)

(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(G02840 IS A POWER-SAW)
(I UNDERSTAND)

(***.     JOHN OWNS A CHAIN)

(I UNDERSTAND)

(***.     DICK HAS A CHAIN)

(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (OWNS))
(I UNDERSTAND)

(***.     THE CUCKOO-CLOCK HAS A CHAIN)

(THE ABOVE SENTENCE IS AMBIGUOUS ** PLEASE RE-PHRASE IT)
```

FIGURE 6 (Cont.)

c. STREAMLINING LINKAGES

```
(***.     JOHN IS A PERSON)

(I UNDERSTAND)

(***.     JOHN IS A TECH-MAN)

(I UNDERSTAND)

(***.     JOHN IS A BOY)

(I UNDERSTAND)

(***.     JOHN IS A STUDENT)

(I UNDERSTAND)

(***.     JOHN IS A BRIGHT-PERSON)

(I UNDERSTAND)

(***.     EVERY BOY IS A PERSON)

(I UNDERSTAND)

(***.     EVERY TECH-MAN IS A PERSON)

(I UNDERSTAND)

(***.     EVERY TECH-MAN IS A BRIGHT-PERSON)

(I UNDERSTAND)

(***.     EVERY TECH-MAN IS A STUDENT)

(I UNDERSTAND)

(***.     EVERY BRIGHT-PERSON IS A PERSON)

(I UNDERSTAND)

(***.     EVERY STUDENT IS A BRIGHT-PERSON)

(I UNDERSTAND)

(***.     EVERY STUDENT IS A PERSON)

(I UNDERSTAND)

END OF EVALQUOTE, VALUE IS ..
(NO MORE INPUT SENTENCES)



 FUNCTION   EVALQUOTE   HAS BEEN ENTERED, ARGUMENTS..
STREAMLINE
(JOHN)

(I FORGET THE MEMBER-ELEMENTS RELATIONS BETWEEN PERSON AND JOHN)
(I FORGET THE MEMBER-ELEMENTS RELATIONS BETWEEN STUDENT AND JOHN)
(I FORGET THE MEMBER-ELEMENTS RELATIONS BETWEEN BRIGHT-PERSON AND JOHN)
(I FORGET THE SET-INCLUSION RELATION BETWEEN PERSON AND TECH-MAN)
(I FORGET THE SET-INCLUSION RELATION BETWEEN BRIGHT-PERSON AND TECH-MAN)
(I FORGET THE SET-INCLUSION RELATION BETWEEN PERSON AND STUDENT)

END OF EVALQUOTE, VALUE IS ..
NIL
```

FIGURE 6 (Cont.)

require it to identify those instances in which specific information and general information differ; the program could then express its amusement at such paradoxes.

b) <u>Resolving ambiguities:</u> The criteria used by the program to decide whether "has," in the format "x has y," should be interpreted "has as parts" or "owns" are the following:

1) Let P be the proposition, "either $\underline{y}$ is known to be part of something, or $\underline{y}$ is an element of some set whose elements are known to be parts of something."

2) Let N be the proposition, "either $\underline{y}$ is known to be owned by something, or $\underline{y}$ is an element of some set whose elements are known to be owned by something."

3) If $P \wedge \sim N$, assume "has" means "has as parts."

If $\sim P \wedge N$, assume "has" means "owns."

If $\sim P \wedge \sim N$, give up and ask for re-phrasing.

4) Let P' be the proposition,

$(\exists u)[[[\underline{y}$ is known to be part of $\underline{u}] \vee [\underline{y}$ is an element of some set whose elements are known to be parts of the elements of $\underline{u}]] \wedge$ $(\exists w)[[u \in w \vee u \subset w] \wedge [x \in w \vee x \subset w]]]$.

5) Let N' be the proposition,

$(\exists u)[[[\underline{y}$ is known to be owned by $\underline{u}] \vee [\underline{y}$ is an element of some set whose elements are known to be owned by the elements of $\underline{u}]] \wedge$ $(\exists w)[[u \in w \vee u \subset w] \wedge [x \in w \vee x \subset w]]]$.

6) If $P' \wedge \sim N'$, assume "has" means "has as parts."

If $\sim P' \wedge N'$, assume "has" means "owns."

Otherwise, give up and ask for re-phrasing.

These criteria are simple, yet they are sufficient to enable the program to make quite reasonable decisions about the intended purpose in various sentences of the ambiguous word "has." Of course, the program can be fooled into making mistakes, e.g., in case the sentence, "Dick has a chain," had been presented before the sentence "John owns a chain," in the above dialogue; however, a human being exposed to a new word in a similar situation would make a similar error. The point here is that it is feasible to automatically resolve ambiguities in sentence meaning by referring to the descriptions of the words in the sentence -- descriptions which can automatically be created through proper prior exposure to unambiguous sentences.

c) Streamlining linkages: All question-answering (model-searching) functions which involve references to set-inclusion or set-membership relations must "know" about the basic properties of those relations, i.e., those functions must have built into them the ability to apply theorems like

$x \subset y \wedge y \subset z \Rightarrow x \subset z$ and

$\alpha \in x \wedge x \subset y \Rightarrow \alpha \in y$;

otherwise the functions would not be able to make full use of the usually limited information available in the form of explicit links. On the other hand, since the functions involved will be "aware" of these theorems, then the set of questions which can be answered is independent of the presence or absence of explicit links which provide the information to the right of the "$\Rightarrow$", provided the information to the left of the "$\Rightarrow$" is available.

The "STREAMLINE" operation starts with the object x which is its argument, and considers all objects linked to x, directly or indirectly, through set-inclusion or set-membership. All explicit links among these objects which can also be deduced by use of the above known theorems are deleted. A response of the form "(I FORGET THE SET-INCLUSION RELATION BETWEEN y AND z)" indicates that whatever links were created by some sentence of a form similar to "(EVERY z IS A y)" are being deleted, and the space they occupied is being made available for other use.

In the above example, the STREAMLINE operation deleted more than half the existing links, at no reduction in the question-answering power of the system. However, the time required to obtain answers to certain questions was significantly increased.

Chapter VI: Formalization and Generalization of SIR

The present version of the SIR system not only demonstrates the possibility of designing a computer which "understands"; it also points the way toward more general, practical systems by providing a useful data representation (the model) and by suggesting useful general information retrieval mechanisms.

SIR's abilities were illustrated by Fig. 1 and, in greater detail, by the conversations of Fig. 5. Unfortunately, the system is quite limited in the number of semantic relations it can "understand" and in the depth of its apparent understanding of any one relation. Moreover, the present system has some basic features which make these limitations extremely difficult to overcome.

The purposes of this chapter are to identify those features which make SIR difficult to extend; to point out how those difficulties arose and how they may be overcome; and to propose a formalism and a computer implementation for a more general semantic information retrieval system which has most of the advantages of SIR but few of its limitations.

The SIR treatment of restricted natural language was discussed at length in Chapter IV and is not of concern here. This chapter deals only with the action of SIR on relational statements which precisely define the desired information storage or retrieval operations.

A. Properties and Problems of SIR.

Let us now examine the present structure and mode of operation of SIR. In particular, we are interested in learning why SIR cannot be extended in simple ways to handle a greater quantity and complexity of information.

1) <u>Program organization:</u> The present computer implementation of SIR is an interdependent collection of specially designed subprograms. Each different information storage or retrieval operation is controlled by a different subprogram.

Such a diffuse program structure has a certain advantage for producing early results with a new experimental system. SIR was primarily developed as an experimental vehicle through which one may learn the best forms of information representation and the best storage and retrieval procedures. As an experimental device, SIR must be easily amenable to changes in its structure and modes of operation. The programmer must be able to learn the most useful interpretations of relational statements and the most useful responses the system should make. This learning takes place as he tries, by means of <u>ad hoc</u> changes to the program, different interpretations and different response modes. These program changes are easiest to make if the program consists of many separate subprograms without much overall structure.

As such a system grows more complicated, each change in a subprogram may affect more of the other subprograms. The structure becomes more awkward and more difficult to generalize as its size increases. Finally, the system may become too unwieldy for further experimentation. (SIR is presently close to this point of diminishing returns.)

However, by the time this barrier is reached many fruitful results may have been attained. <u>Ad hoc</u> features may coalesce into general principles. Desirable features may be discovered, and uniform methods may emerge for handling problems which originally seemed quite different from each other. In particular, my experiences in developing SIR to its present state have enabled me to specify the more uniform, more general, more powerful system proposed in Sections B and C below.

2) The model: The model is a flexible body of data whose content and organization are crucial factors in SIR's learning and question-answering abilities. SIR's "knowledge" is derived from two sources: facts represented in the model, and procedures embodied in the program. Basic procedures in the program provide for automatic revision of the model, if necessary, whenever new information is presented to the system. No such automatic procedures exist for revising the program itself.

The greater the variety of information which can be stored in the model, the more flexible the resulting system is; the more specific requirements and restrictions which are built into the program, the more rigid and less general the overall system is. It seems desirable, then, to store in the model a great variety of information, including facts about objects, relations, and the operation of the program itself. The program would then consist simply of storage procedures which would modify the model, and retrieval procedures whose actions would be controlled by data in the model. The user could then simply "tell" the system how to change its retrieval procedures, whenever such changes are desired.

Such a flexible system, whose program is "driven" by the model, is an ultimate objective of this research. Unfortunately, this objective must be approached by successive approximations. A model-controlled system cannot be designed at the outset for the following reasons:

a. In order to store all the significant, controlling information in the model, we must first discover what constitutes the significant information in a semantic information retrieval system. After developing any workable program-plus-model system we are in a better position to recognize truly important features and to transfer control of them to the model.

b. The value and efficiency of the system depends upon the structure of the model, and the manner in which the program and model interact. One should limit the complexity of the model until the organization of the model and of the overall system have been proved feasible.

c. The problem of how to express controlling information which we wish to add to the model, e.g., how best to describe search and deduction procedures, must be solved along with the problems of representing and utilizing that information once it is in the model. Formalisms for describing such control procedures are easier to devise after some experience has been gained in the use of similar procedures. This experience, in turn, is easy to develop through experimentation with the program portion of simplified semantic information retrieval systems.

In SIR the model consists only of descriptions of objects and of classes. The number, kind, and interpretation of the descriptors (attributes) in the model is determined by the program. The information about how the meanings of certain attributes are related to each other is incorporated in the subprograms which identify those attributes, rather than in the model.

Although SIR is approaching its limit in usefulness, experience with the system has brought me to the point where I can confidently propose an improved, generalized system. The system proposed in sections B and C below keeps the now proven description-list organization for the model; it increases the variety of data to be stored in the model; it transfers some of the information about the attributes from the program to the model; and it provides the user with a simplified method for experimenting with the deductive procedures of the system.

3) Question-answering method: In order to describe how SIR's question-answering behavior has been achieved and how it can be improved, I must first introduce some notation. As described in Section III.D.3, each relation in the SIR system is a dyadic relation and hence is represented in the model by two attribute links. Table a. gives the correspondence between relation names and attribute names, and a typical English interpretation for each relation. Note that I use the familiar infixes "$\subset$" and "$\in$" for set-inclusion and set-membership, respectively, although functional notation, *e.g.*, "equiv[x;y]," is used for all other relations. Also, the usual symbols of mathematical logic, which are defined in Appendix I, will be used below when convenient.

A relation "holds" for specified arguments, *i.e.*, a relation with specified arguments (called a *predicate*) is "true," if and only if any reasonable English interpretation of the relational statement is a true English statement. An English interpretation should be considered "reasonable" only if the natural-language processing part of the system would translate it into the given relational statement. A relation with specified objects as arguments clearly is true if the objects are linked in the model by the attributes which correspond to the relation. However, frequently such a predicate is "true" even when its arguments are *not* directly linked. In such cases the truth

Relation	Attribute on property-list of x	Attribute on property-list of y	Typical English interpretation
x⊂y	SUPERSET	SUBSET	An x is a y.
x∈y	MEMBER	ELEMENT	x is a y.
equiv[x;y]	EQUIV	EQUIV	x and y name the same object.
owng[x;y]	OWNED-BY-EACH	POSSESS-BY-EACH	Every y owns an x.
own[x;y]	OWNED	POSSESS	y owns an x.
partg[x;y]	SUPERPART-OF-EACH	SUBPART-OF-EACH	An x is part of a y.
part[x;y]	SUPERPART	SUBPART	An x is part of y.
right[x;y]	LEFT	RIGHT	The x is to the right of the y.
jright[x;y]	JLEFT	JRIGHT	The x is just to the right of the y.

Table a: RELATIONAL NOTATION

of the predicate can be determined indirectly from other information available in the model or in the program.

SIR contains a separate subprogram for determining "truth" for each relation in the system. These are the subprograms responsible for answering "yes-or-no" questions. For example, the answer to the question, "is the chair to the right of the table?" would be found by a subprogram called "rightq" which deals with the truth of the "right" relation. "Chair" and "table" would be the inputs to the "rightq" program, which would then search the model and make an appropriate response.

During the development of SIR, procedures for establishing the truth of relations had to be explored independently for each relation and so a separate program was written for each relation. The detailed operation of these subprograms was described in Chapter V. Now, as we consider how to generalize the system, the time has come to look for common features of these subprograms. Such common features could serve as the basis for a simpler, more unified program structure. Indeed, such common features <u>have</u> been found, and they are exploited in the general system to be described in Sections B and C below.

The first step in trying to simplify the truth-testing procedures is to express the procedures in such a way that their operations can easily be compared and understood. In practice each of the truth-testing subprograms operates by searching the model, looking for certain combinations of attribute links. However, since the existence of an attribute link implies the truth of a corresponding predicate, we may consider the subprogram as deducing the truth of a predicate from the fact that certain other predicates are true. Such deduction procedures are conveniently expressible in the first-order predicate calculus (the "quantificational calculus").

Frequently the truth of a predicate depends upon the fact that the <u>relation</u> involved has a special property, <u>e.g.</u>, transitivity. These properties of relations may conveniently be described by "definition" statements in which a bound variable stands for the name of some unspecified relation. These definitions are simply abbreviations which will become ordinary quantificational calculus statements when the bound variables are replaced by particular relation names.

The properties defined below are useful for describing some of the SIR relations:

Symmetry: $\mathcal{S}(P) =_{df} (\forall x)(\forall y)[P[x;y] \Rightarrow P[y;x]]$

Reflexivity: $\mathcal{R}(P) =_{df} (\forall x)[P[x;x]]$

Transitivity: $\mathcal{T}(P) =_{df} (\forall x)(\forall y)(\forall z)[P[x;y] \wedge P[y;z] \Rightarrow P[x;z]]$

The following logical sentences hold throughout SIR and represent basic properties of the "equiv" relation:

$(\forall P)(\forall x)(\forall y)(\forall z)[P[x;y] \wedge equiv[x;z] \Rightarrow P[z;y]]$

$(\forall P)(\forall x)(\forall y)(\forall z)[P[x;y] \wedge equiv[y;z] \Rightarrow P[x;z]]$

Table b. lists predicate calculus statements corresponding to the deduction procedures actually used in the SIR subprograms for truth-testing. These statements were obtained by studying the SIR subprograms, and they accurately represent the operation of those subprograms except for the following:

a. All quantifiers range over only the finite universe of objects, classes, and relations represented in the model.

b. Each subprogram contains built-in mechanisms for searching the model in the course of trying to apply one of the deduction procedures. The linkage structure of the model allows the programs to make direct, exhaustive searches through just the relevant portions of the model.

c. When alternative deduction procedures are available for testing a predicate, each subprogram specifies the order in which the procedures should be attempted. As is illustrated by the "Exception Principle" (Section V.B.1), the use of alternate deduction procedures may result in different answers to a question. This means that, from a purely predicate-calculus point of view, the deduction procedures together with the information stored in the model may form an inconsistent system. Therefore the order in which deduction procedures are used influences the answers obtained. In the present form of SIR the ordering rule has been that those procedures dealing with indirect links are to be used only if no answer can be obtained by using those procedures dealing with more direct links.

d. Each subprogram is independent and contains complete programs for its deduction procedures. Since some of the deduction procedures in different subprograms are similar, some program segments appear several times in the SIR system. For example, programs which test whether a particular class-inclusion relation holds appear in most of the truth-testing subprograms. This program redundancy results from the independent subprogram organization of SIR and should be removed in a more uniform system.

Relation being tested	Deduction Procedures*
$\subset$	1. $\mathcal{T}[\subset]$
	2. $x{=}y \Rightarrow x \subset y$
	3. $equiv[x;y] \Rightarrow x \subset y$
$\in$	4. $\alpha \in x \wedge x \subset y \Rightarrow \alpha \in y$
equiv	5.,6.,7. $\mathcal{T}[equiv], \mathcal{R}[equiv], \mathcal{S}[equiv]$
owng	8. $\sim owng[x;x]$
	9. $owng[x;y] \wedge z \subset y \Rightarrow owng[x;z]$
	10. $owng[x;y] \wedge x \subset z \Rightarrow owng[z;y]$
own	11. $own[x;y] \wedge x \in z \Rightarrow own[z;y]$
	12. $owng[x;y] \wedge z \in y \Rightarrow own[x;z]$
partg	13. $\sim partg[x;x]$
	14. $partg[x;y] \wedge z \subset y \Rightarrow partg[x;z]$
part	15. $part[x;y] \wedge x \subset z \Rightarrow part[z;y]$
	16. $part[x;y] \wedge partg[z;x] \Rightarrow part[z;y]$
	17. $partg[x;y] \wedge z \in y \Rightarrow part[x;z]$
right, jright	18. $right[x;y] \Rightarrow \sim right[y;x]$
	19. $\mathcal{T}[right]$
	20. $jright[x;y] \Rightarrow right[x;y]$
	21. $jright[z;y] \wedge z \neq x \Rightarrow \sim jright[x;y]$
	22. $jright[x;w] \wedge w \neq y \Rightarrow \sim jright[x;y]$
	23. $right[x;y] \wedge right[y;z] \Rightarrow \sim jright[x;z]$

Table b: DEDUCTION PROCEDURES IN SIR SUBPROGRAMS

* Universal quantification over all free variables is assumed.

Thus far I have been discussing only those programs which answer "yes-or-no" questions. More complex questions, such as "Where is the table?" and "How many fingers does John have?", require different question-answering procedures. SIR contains an additional subprogram for each of these complex question forms. These subprograms will be discussed further in Paragraph C.3 below.

B. Formalism for a General System.

Given a suitable formal system, a separate truth-testing subprogram for each relation in the SIR system would not be necessary. Instead, a single "proof-procedure" program could serve for answering all "yes-or-no" questions.

The deduction procedures of Table b, could be used as the axioms of such a formal system. However, the study of these "axioms" has suggested an alternative system which is more concise, more intuitively meaningful, and easier to extend to new relations. This alternative formal system is the subject of this section.

1) Interactions. Two relations "interact" if, in order to test the truth of a predicate involving one of the relations, it is necessary first to test the truth of some predicate involving the other. Whenever two or more relations appear in the same deduction-procedure statement in Table b, we may say that those relations interact.

Interactions may be classified informally as follows:

a. Interactions between the $\in$ or $\subset$ relation and some other relation.

b. Interactions between relations whose meanings are similar to each other. (This "similarity" will be defined more precisely in Section 2 below.)

c. Interactions which arise principally because of some peculiarity of <u>one</u> of the relations involved.

d. Other interactions.

Interactions are of interest because they create the biggest obstacle to generalizing the SIR system. Whenever a new relation is added to the system, the programmer must identify all the relations in the system which interact with the new relation, and modify the system to allow for the interactions. With the present system, this means modifying each of the question-answering subprograms associated with the interacting relations. This formidable reprogramming task accounts for the fact that the deduction schemes in the present version of SIR do not allow for <u>all</u> the intuitively necessary interactions between relations in the system. For example, if SIR is told that an <u>x</u> is part of every <u>y</u> and that <u>z</u> owns a <u>y</u>, it cannot deduce that <u>z</u> owns an <u>x</u>. To perform this and similar deductions SIR would have to "know" about additional interactions among the relations <u>part</u>, <u>partg</u>, <u>own</u>, <u>owng</u>, <u>∈</u>, and <u>⊂</u>.

Almost all the interactions accounted for in the present system and in the deduction procedures of Table b. are of type "a," "b," or "c," according to the above classification scheme, <u>i.e.</u>, they involve the relations ∈ or ⊂, relations whose meanings are similar, or relations with individual peculiar properties. The formal system to be described below will eliminate the need for explicitly considering <u>any</u> interactions of these three types. Once a new relation is properly described according to simple, intuitive rules, any type "a," "b," or "c" interactions between it and other relations will automatically be accounted for by the logical system. Although other (type "d") interactions may still exist, they will be easy to describe and modify. For example, a single simple statement will be sufficient to make the system "aware" of the interaction between part-whole and ownership relations illustrated in the previous paragraph.

2) <u>SIR1: A proposed formal system for truth-testing:</u> The formal system called "SIR1" to be proposed here will consist of: definitions of certain terms, including terms which describe strings of symbols; a standard interpretation for the symbols; and a logical method for determining whether certain strings called "sentences" of SIR1 are "true." The significance of the system is that all "yes-or-no" questions which can be answered by SIR, and a great many which cannot, are expressible as sentences in SIR1; *i.e.*, the standard interpretation of a formal sentence is its corresponding English question. Further, if a sentence is "true" in SIR1, then the answer to its corresponding question is "yes." These points will be illustrated by examples below. A computer implementation of SIR1 will be discussed in Section C of this chapter.

a. Definitions:

<u>basic object</u> =df any object which is described in the model and which has the following property: No object described in the model may be related to a basic object by being a member or a subset of it.

<u>basic relation</u> =df a symbol which names a relation whose arguments must all be basic objects.

<u>variable</u> =df a symbol used in place of the name of some unspecified object described in the model. The standard interpretation of the name of an object is, of course, the object itself.

<u>basic predicate</u> =df a basic relation written as a function of the names of basic objects or of variables which stand for the names of basic objects. The standard interpretation of a predicate is that the specified relation holds between the specified objects.

$\in$-quantifier =df either of the symbols "$(\forall v_1 \in v_2)$" or "$(\exists v_1 \in v_2)$," where v_1 is any variable and v_2 is any variable, any object name, or the special symbol "M" which stands for "model." These $\in$-quantifiers are related in the first-order predicate calculus as follows:

$$(1)\quad \begin{aligned} &(\forall \alpha \in x)[R[\alpha]] \text{ =df } (\forall \alpha \in M)[\alpha \in x \Rightarrow R[\alpha]] \\ &(\exists \alpha \in x)[R[\alpha]] \text{ =df } (\exists \alpha \in M)[\alpha \in x \wedge R[\alpha]] \end{aligned}$$

where $(\forall \alpha \in M)$ and $(\exists \alpha \in M)$ are the usual universal and existential quantifiers of mathematical logic, respectively, except for an explicit reminder that they range over only the finite universe of objects described in ~~the model; and R[α] is any predicate, although~~ it usually contains at least one occurrence of the symbol α among its arguments.

An $\in$-quantification of a string S is the string "Q[S]" where Q is any $\in$-quantifier. The first variable in Q is then called bound by the $\in$-quantification of S for all its occurrences in Q and in S, including occurrences as the second variable of other $\in$-quantifiers.

A link-predicate is defined recursively as follows:

i) A basic predicate is a link-predicate.
ii) The strings "$v_1 \in v_2$" and "$v_1 = v_2$," where v_1 and v_2 are any object-names or variables, are link predicates.
iii) An $\in$-quantification of a link-predicate is a link-predicate.

Link-predicates may be used to represent most of the relations which are represented by attribute links in the present version of SIR.

A well-formed-formula (wff) is defined recursively as follows:

i) A link-predicate is a wff.
ii) Any propositional function of wff's is a wff.
iii) Any $\in$-quantification of a wff is a wff.

An occurrence of a variable in a wff is called free if the occurrence is not bound by an $\in$-quantification of some string containing that occurrence.

A sentence =df a wff which contains no free variables.

An object-predicate =df a wff which contains exactly one free variable.

b. Logical system:

The axioms of SIR1 are sentences which, under standard interpretation, describe properties of individual basic relations and specify type-"d" interactions between basic relations.

Any sentence in SIR1 can be transformed into a sentence in the standard first-order predicate calculus (the "quantificational calculus") by putting each ∈-quantifier into its "∈M" form by use of the equations (1), and then omitting the "∈M." All the usual deduction procedures of the quantificational calculus are acceptable deduction procedures in SIR1. Therefore, any theorem provable from SIR1 axioms in the quantificational calculus is also a theorem of SIR1; i.e., it is a "true" sentence of SIR1, provided "∈M"s are inserted into all quantifiers, regardless of the state of the current model. In other words, SIR1 is reducible to the quantificational calculus. This reducibility provides us with methods -- namely the methods of quantificational calculus, such as Subordinate Proof Derivation ("Natural Deduction") -- for proving whether sentences of SIR1 are theorems. However, we need different, more direct methods for testing the truth of SIR1 sentences which depend on the model. These truth-testing methods must be implemented on the computer, for they constitute the basic question-answering mechanism of the generalized semantic information retrieval system. However, I shall first describe a totally impractical truth-testing method which demonstrates that a decision procedure exists for testing "truthhood" of SIR1 sentences with respect to particular SIR1 model. A more efficient, heuristic approach will be described in paragraph C.2 below.

The SIR1 model is quite similar to the SIR model. It consists of a finite number of object names, each of which is "described" by a finite list of attribute-value pairs. Each attribute may name an object-predicate which is true of the described object, or it may be a link which relates the described object to another object. This latter object is named in the value corresponding to the given attribute. In Section C I shall describe the nature of SIR1 attributes more precisely. For present purposes it is sufficient to assume that the information carried by each attribute on a property-list in the SIR1 model can be expressed in some well-defined way as a SIR1 sentence.

A SIR1 sentence is considered "true" if the sentence can be deduced from the SIR1 axioms and the information in the SIR1 model. A decision procedure for this deduction follows:

i) For each attribute in the model, write the SIR1 sentence which expresses the same thing.

ii) Let A = the conjunction of all the sentences found in i) and of all the SIR1 axioms. Consider the sentence

(2) $A \Rightarrow S$

where S is the sentence being tested.

iii) Put all ϵ-quantifiers in (2) into the "ϵM" form by using equations (1).

iv) Let $o_1, o_2, \ldots, o_n$ be the names of the objects described in the model. Eliminate the quantifiers in (2) by replacing each string of the form $(\forall v \epsilon M)[R[v]]$, where v is any variable and R is any predicate possibly depending on v, with the finite conjunction

$$R[o_1] \wedge R[o_2] \wedge \ldots \wedge R[o_n];$$

and by replacing each string of the form $(\exists v \epsilon M)[R[v]]$ with the disjunction

$$R[o_1] \vee R[o_2] \vee \ldots \vee R[o_n].$$

v) Test the resulting expression by a decision procedure for the propositional calculus, *e.g.*, by truth-table analysis. *S* is true with respect to the model, and the question corresponding to S should be answered "YES," if and only if this final expression is a theorem of the propositional calculus.

c. Examples and comments:

i) Object-predicates: As defined above, an object-predicate is a SIR1 wff which contains exactly one free variable. If that free variable is replaced by an object-name, the object-predicate becomes a

SIR1 sentence. The standard interpretation of an object-predicate applied to an object in the SIR1 model is that the sentence obtained by replacing the free variable in the predicate by the object-name is a true sentence. This resulting sentence may then be used as an additional axiom in any SIR1 logical deduction procedure.

Object-predicates may be placed on the property-list of any object in the SIR1 model. Their purposes are to describe those properties of the object which cannot easily be expressed, in terms of link-predicates, as specific associations with other objects.

ii) Basic relations: The "ϵ" relation occupies a special place in SIR1 because of its connection with ϵ-quantifiers, and is treated in the formalism as if it were a basic relation. The identity relation "=" is also treated as a basic relation because identity is a useful feature to have in a logical system based on the quantificational calculus. The SIR relation "equiv" was simply an equivalence relation used to identify when different object-names referred to the same object. In SIR1 it is sufficient to subsume the function of "equiv" under the "=" sign; *i.e.*, the formal statement "x=y" is considered to be true if either *x* and *y* are the same symbol, or if "equiv[x;y]" is a true predicate in the SIR model.

The predicates in Table c_1 show the basic relations and the object predicate needed by SIR1 in order to deal with all the relations covered by SIR programs.

iii) Connections between SIR and SIR1 relations: Table c_2 lists a SIR1 expression which should be used in place of each SIR predicate. Corresponding expressions have exactly the same interpretations; the SIR1 statements are more complicated, but they utilize

Predicate	Standard Interpretation
$x \in y$	*x* is a member of the set *y*.
$x = y$	Either *x* and *y* are identical, or they are two names for the same object.
ownb[x;y]	*x* is owned by *y*.
partb[x;y]	*x* is part of *y*.
rightb[x;y]	*x* is to the right of *y*.
jrightb[x;y]	*x* is just to the right of *y*.
single[x]	$(\exists\alpha\in M)[\alpha\in x \wedge (\forall\beta\in M)[\beta\in x \Rightarrow \beta=\alpha]]$ (interpretation: *x* has exactly one member.)

Table c_1: BASIC RELATIONS OF SIR1

....

SIR Predicate	SIR1 Expression
$x \subset y$	$(\forall\alpha\in x)[\alpha\in y]$
$x \in y$	$x \in y$
equiv[x;y]	$x = y$
owng[x;y]	$(\forall\beta\in y)(\exists\alpha\in x)[\text{ownb}[\alpha;\beta]]$
own[x;y]	$(\exists\alpha\in x)[\text{ownb}[\alpha;y]]$
partg[x;y]	$(\forall\beta\in y)(\exists\alpha\in x)[\text{partb}[\alpha;\beta]]$
part[x;y]	$(\exists\alpha\in x)[\text{partb}[\alpha;y]]$
right[x;y]	$(\exists\alpha\in x)(\exists\beta\in y)[\text{rightb}[\alpha;\beta]] \wedge \text{single}[x] \wedge \text{single}[y]$
jright[x;y]	$(\exists\alpha\in x)(\exists\beta\in y)[\text{jrightb}[\alpha;\beta]] \wedge \text{single}[x] \wedge \text{single}[y]$

Table c_2: SIR PREDICATES EXPRESSED IN SIR1 fewer basic symbols and they show more logical structure than their SIR counterparts.

The SIR1 link-predicate corresponding to "partg[x;y]" in Table c_2 has the interpretation, "Some $\underline{x}$ is part of every $\underline{y}$." Although this is the interpretation used in most SIR question-answering subprograms, "partg[x;y]" might equally well be interpreted, "Every $\underline{x}$ is part of some $\underline{y}$," in which case the SIR1 link-predicate $(\forall\alpha\epsilon x)(\exists\beta\epsilon y)[\text{partb}[\alpha;\beta]]$ should be used. Actually the interpretation of "partg[x;y]" suggested in Table a., "An $\underline{x}$ is part of a $\underline{y}$," is ambiguous. This ambiguity occurs because the natural-language input system in the present version of SIR cannot discover the finer meanings of "An $\underline{x}$ is part of a $\underline{y}$." Perhaps the most suitable representation for this latter sentence is a conjunction of two SIR1 link-predicates

$$(\forall\beta\epsilon y)(\exists\alpha\epsilon x)[\text{partb}[\alpha;\beta]]\wedge(\forall\alpha\epsilon x)(\exists\beta\epsilon y)[\text{partb}[\alpha;\beta]]$$

The SIR predicate "right[x;y]" was interpreted as "The $\underline{x}$ is to the right of the $\underline{y}$." This English sentence implies first that $\underline{x}$ and $\underline{y}$ are each sets containing unique elements, and secondly that those elements bear a certain positional relationship to each other. In SIR the special subprogram "specify" was used to determine the nature of the sets involved, before the positional information was considered. Similarly, the SIR1 expression must be the conjunction of the object-predicates "single[x]" and "single[y]" to describe the special nature of $\underline{x}$ and $\underline{y}$, and the link-predicate whose interpretation is, "an $\underline{x}$ is to the right of a $\underline{y}$." Similarly, object-predicates, as well as a link-predicate, are needed to represent the SIR "jright" relation.

iv) __Axioms of SIR1__: Some useful properties of SIR1 relations are defined as follows:

P is symmetric:
$\mathcal{S}(P) =_{df} (\forall x\in M)(\forall y\in M)[P[x;y] \Rightarrow P[y;x]]$

P is asymmetric:
$\bar{\mathcal{S}}(P) =_{df} (\forall x\in M)(\forall y\in M)[P[x;y] \Rightarrow \sim P[y;x]]$

P is reflexive:
$\mathcal{R}(P) =_{df} (\forall x\in M)[P[x;x]]$

P is set-nonreflexive:
$\bar{\mathcal{R}}(P) =_{df} (\forall x\in M) \sim(\forall \beta\in x)(\exists \alpha\in x)[P[\alpha;\beta]]$

P is transitive:
$\mathcal{T}(P) =_{df} (\forall x\in M)(\forall y\in M)(\forall z\in M)[P[x;y] \wedge P[y;z] \Rightarrow P[x;z]]$

P is uniquely linked:
$\mathcal{U}(P) =_{df} (\forall x\in M)(\forall y\in M)[P[x;y] \Longrightarrow (\forall \alpha\in M)[[\alpha\neq y \Rightarrow \sim P[x;\alpha]] \wedge [\alpha\neq x \Rightarrow \sim P[\alpha;y]]]]$

Notice that these properties will be expressed by ordinary SIR1 sentences when the bound variable "P" is replaced by the name of a SIR1 relation.

Table d. is a list of all the axioms necessary to give SIR1 at least the question-answering ability of the SIR deduction procedures in Table b, except for the "axioms" derived from object predicates on the property-lists of particular objects. In Table b, deduction procedures no. 1-4, 9-11, 14, and 15 all represent interactions with the "$\in$" or "$\subset$" relations, i.e., type "a" interactions. Corresponding axioms are not needed in SIR1 because of the way "$\subset$" is defined (see Table c_2) and the way $\in$-quantifiers are used. Table b. no. 12 and 17 are interactions between "similar" relations, i.e., type "b" interactions. "Similar" relations are those which are defined in terms of a single basic relation in SIR1. Additional axioms are not needed because information about interactions between "similar" relations are implicit in their definitions as link-predicates. Procedure no. 16 is really a statement of the transitivity of the basic part-whole relation (a type "c" interaction), somewhat obscured by a statement of the interaction between the similar "part" and "partg" relations

Axioms	Discussion
$\mathcal{T}(=)$ $\mathcal{A}(=)$ $\mathcal{I}(=)$	**This fact that "=" is an equivalence relation is not strictly necessary in the axioms, since it is built into the logical system.**
$\mathcal{A}$(ownb) $\mathcal{R}$(partb)	**cf. no. 8 and 13, Table b. These are "experimental" axioms, which should be dropped from the system if too many exceptions turn up.**
$\mathcal{T}$(partb)	**cf. no. 16, Table b.**
$\mathcal{I}$(rightb)	**cf. no. 18, Table b.**
$\mathcal{T}$(rightb)	**cf. no. 19, Table b.**
$\mathcal{U}$(jrightb)	**no. 21 and 22, Table b., were needed because this property was missing.**
$(\forall x\in M)(\forall y\in M)[\text{jrightb}[x;y] \Rightarrow \text{rightb}[x;y]]$	**cf. no. 20, Table b.**
$(\forall x\in M)(\forall y\in M)(\forall z\in M)[\text{rightb}[x;y] \wedge \text{rightb}[y;z] \Rightarrow \sim\text{jrightb}[x;z]]$	**cf. no. 21, Table b.**
	The last two axioms represent true type "d" interactions between rightb and jrightb.

Table d: SIR1 AXIOMS

(a type "b" interaction). Interactions 21 and 22 of Table b. are of type "c," for they are due solely to the peculiar property of "jright" which is expressed in SIR1 by $\mathcal{U}$(jrightb). Finally, no. 20 and 23 of Table b. are true type "d" interactions, and corresponding axioms are necessary in SIR1.

Let me now make this discussion more precise. The deductive systems of SIR and SIR1 are both based on the quantificational calculus. The only difference between them is that the SIR deduction procedures, in Table b., are a description of the operation principles of an existing computer program. SIR1 is a formally developed system which may eventually contribute to the specification for a computer program. If the SIR1 system with its short list of axioms (Table d.) is already as effective a "yes-or-no" question-answerer as the programs described by the SIR procedures in Table b., then adding those procedure rules to SIR1 cannot increase the power of SIR1. In other words, SIR1 must already contain all the information available in the rules of Table b. To prove that this is indeed the case, I have shown that SIR1 sentences corresponding to each of the rules of table b. are <u>theorems</u> in SIR1. The method used was to reduce the SIR1 axioms and sentences to the quantificational calculus and then to prove the theorems by Subordinate Proof Derivations (Appendix I). The details are given in Appendix II.

v) <u>ϵ-quantifiers:</u> The most obvious difference between SIR1 and the quantificational calculus is the occurrence in SIR1 of ϵ-quantifiers. These new symbols serve three functions, the most obvious but least important of which is notational conciseness. Since the value of any notational device depends upon its understandability, ϵ-quantifiers are valuable because they indicate the intended interpretation of SIR1 sentences to the user or reader. Finally, ϵ-quantifiers are important for the computer implementation of SIR1. They are indicators which relate the formal system to particular model search-procedures. Details of a proposed implementation scheme are presented in Section C.

C. Implementation of the General Question-Answering System.

A semantic information retrieval system which can be as effective as SIR and yet have the uniformity and generality of the SIR1 formalism must have the following components:

i) a <u>model</u> patterned after the SIR model but containing more complete information in its linkages and containing a larger class of describable objects.

ii) a <u>theorem-proving program</u> which can determine whether certain assertions are true, on the basis of axioms of SIR1 and current information in the model.

iii) a <u>programming language</u> for specifying question-answering procedures which are more complex than truth-testing.

In addition, these components must be designed to work together to form a compact, efficient system. A detailed description of each of these components of the proposed system will follow shortly.

A program to translate natural or restricted English into formal relational terms, and a program to annex new relational information to the model, are also necessary components of any semantic question-answering system. The latter annexing program is straight-forward and all the basic mechanisms are already available in SIR. English translation is a linguistic problem whose detailed study is beyond the scope of this paper. The trivial format-matching solution (Chapter IV) may be used until something better becomes available. In any case, I shall assume the availability of some mechanism for accepting new information in a form convenient to the human user, and then inserting corresponding relational information into the model.

1) The model: As discussed in section A.2 above, one objective of this research is to find ways of using information stored in the model to control the operation of the system, since that information can be modified most easily. Since the operation of any theorem-proving program is "controlled" by the axioms of the formal system involved, the axioms for SIR1 should be stored in the model.

The SIR model consists of objects and associated property-lists. The advantage of this model structure is that the program using the model can obtain all the information about an object, such as how it is related to other objects, simply by referring to the object itself. The SIR1 axioms of Table d. all describe either properties of SIR1 basic relations or interactions between basic relations. These axioms should be stored, then, on the property-lists of the basic relations which they affect. In this way the theorem-proving program will be able to find relevant axioms by looking at the property-lists of the basic relations it is concerned with, and the human user or programmer will be able to modify the axiom set by "telling" the system to modify its model, without any reprogramming being necessary. Object-predicates define additional axioms which apply to particular objects. Therefore, they should be stored on the property-lists of the objects involved.

In SIR, a relation between objects is represented in the model by attribute-links on the property-lists of the objects. Each relation is uniquely represented by particular attributes. Simple (types "a" and "b") interactions between relations can not be represented in the model, but rather have to be "known" by the program.

As has been shown, the class of SIR relations roughly corresponds to the class of relations represented in SIR1 by link-predicates. Each link-predicate, in turn, is defined in terms of a SIR1 basic relation. We must now decide how to represent relational information in the SIR1 model.

Each basic relation could be uniquely represented by particular attributes. However, these attributes would not be sufficient to represent all the facts which were representable in SIR. For example, the sentence "Every hand is part of a person," could be represented in SIR1 by locating every object in the system which is a member of the set "hand," and linking each of them to some member of the set "person" with the attributes corresponding to the partb basic relation. However, it is not clear which hands should be parts of which persons; and the general fact concerning hands and persons would be unavailable for future deductions, e.g., when a new individual "person" is introduced into the model.

Alternatively, one could represent each possible link-predicate by a different attribute. The disadvantages of such a scheme would be twofold: First, much of the flexibility introduced by the definition and use of link-predicates would be lost, since special symbols would have to be assigned as attributes for each link-predicate actually used in a model; secondly, the important structure of the link-predicate, i.e., the basic predicate and ϵ-quantifiers of which it is composed, would be undiscoverable except by means of some table look-up or other decoding procedure.

I propose that, corresponding to the attribute-links of SIR, SIR1 should use descriptions of the link-predicates involved. The attribute on the property-list of an object should itself be a property-list. This subproperty-list would contain special attributes whose values were the basic relation involved and the string of ϵ-quantifiers which produce the link-predicate from that basic relation. An additional item on the subproperty-list could identify the argument-position of the described object, thus eliminating the need for more than one symbol (corresponding to the attribute-link symbols of SIR) for each basic relation. With this representation no special symbol assignment or other anticipatory action is necessary in order to add new link-predicates to the model. Any link-predicate recognized by the input program and based on an available basic relation is representable.

The names of object-predicates should be another kind of attribute which may appear on SIR1 property-lists. The object-predicates should themselves by SIR1 objects whose property-lists contain their definitions as SIR1 wff's. In this way object-predicates may easily be défined or applied to new objects.

In summary, the basic objects in the SIR1 model are the words which denote: individuals, classes, basic relations, and object-predicates. A property-list is associated with each basic object. Attributes in the descriptions of individuals and classes are either the names of object-predicates, or themselves property-lists which describe link-predicates. If lists describing link-predicates, the values corresponding to those attributes give the other objects associated with the described object through the described link-predicate. The property-lists of basic relations contain the axioms which specify properties of the described relations. The property-lists of object-predicates contain the definitions of the object-predicates in terms of SIR1 wff's.

2) The Theorem-prover: In paragraph B.2 above I presented a decision procedure for testing the truth of any SIR1 sentence with respect to a given SIR1 model. Unfortunately, that procedure is impractical since it requires the enumeration of every object and every link in the model, and the consideration of every known logical truth in the course of each truth-test. Clearly these procedures would involve an inordinate amount of time. Also, I have gone to great lengths to develop a model structure which enables the system to save time by having information organized and accessible in a convenient way; the above-mentioned decision procedure completely ignores the structure of the model.

Instead of an impractical decision procedure, I propose that SIR1 use a heuristic Theorem-Proving program ("TP") for its truth-testing. TP will start its truth-testing with the most relevant axioms and model linkages, introducing additional facts only when needed. The model structure will dictate what constitutes "most relevant," as will be explained below.

The best example of a heuristic theorem-proving program in Newell and Simon's "Logic Theorist" (LT) (27), a program which proves theorems in the propositional calculus. Since TP will be modeled somewhat after LT, let us consider the general behavior of LT. LT must be given a list of true theorems or axioms, and a statement (the "problem") whose proof is desired. The system tries to prove the test-statement by showing that it, or some statement from which it can easily be deduced, is a substitution instance of a true statement. The true statement must be either a theorem or a statement whose proof is easily obtained from the list of theorems. LT has several methods -- the principal ones called chaining, detachment, and replacement -- for creating statements from which the problem statement can be deduced, and for selecting "relevant" theorems from the theorem list. LT also contains special devices for keeping track of sub-problems and keeping out of "loops."

LT was designed largely as a model of the behavior of naive students of logic, and is reasonable successful as such. It has not been a very effective theorem-prover, partly because its methods and selection heuristics are not powerful enough, and partly because the problem domain -- the propositional calculus -- has a simple decision procedure (46) which makes any alternative approach seem weak. TP must deal with a more complicated problem domain than that of LT. It is concerned with a domain containing a possibly large, although finite, number of objects, relations, and axioms. Also, the objects and relations as well as the axioms may be changed from problem to problem. However, the actual proofs of SIR1 sentences by TP will, on the average, be shorter and simpler than typical LT proofs. After all, TP parallels the human mechanisms for recalling facts in memory and doing some simple reasoning, not for solving formal mathematical problems. Development of elaborate logical ability in a computer must come after the achievement of our present goal: a mechanism for simple, human-like communication. Deductive methods similar to those of LT should be adequate for TP, provided we can provide a mechanism for selecting the "most relevant" true facts from which to start each deduction; and of course the central information organizational device of SIR and SIR1 -- the model -- is just such a mechanism.

Therefore, I propose that TP contain the same deductive methods as LT, and in general be patterned after LT, with the following important exceptions:

a. In trying to apply its methods, LT always scans the complete list of true theorems. TP should initially attempt a proof with a small list of "most relevant" truths extracted from the model. If the proof methods fail, the list of truths should be gradually expanded until the "relevant" portion of the model is exhausted; or, more commonly, until the specified time or effort limits have been reached. One method of generating "relevant" truths for the proof of a SIR1 sentence S is the following:

i) Let B= the set of all basic relations which appear in S. Let F= the set of all object-names in the model which appear in S as arguments of members of B.

ii) Construct a truth list consisting of three parts: those axioms which appear on the description lists of the basic relations in B, those link-predicates which involve relations in B and which are described by attributes of objects in F, and those axioms obtained from object-predicates which appear on the property lists of objects in F.

If a proof cannot be found, the initial truth list can be expanded by enlarging B or F in any of the following ways, and then repeating step ii):

iii) Add the "ϵ" relation to B. This relation is important for deductions which involve transforming or removing ϵ-quantifiers.

iv) Add to B any new basic relations which appear in the current truth list. Whenever basic relations interact, an axiom on the property-list of one will name the other, thereby introducing it into the system. Also, axioms from object-predicates may introduce new basic relations.

v) Add to F all object-names which appear in values of those attributes of objects already named in F, which involve relations already named in B.

Each iteration of step iv) or v) and step ii) will add facts to the truth list which are more indirectly related to the test sentence than any facts previously available. When no new facts can be added in this way, the truth list will contain all the information in the model which may be relevant for the desired proof. However, I expect that in most cases true sentences will be provable from a truth list obtained in very few iterations.

b. SIR1 is concerned with the truth of relational statements with respect to the model, whereas LT is concerned with the universal truth of logical propositions. The ultimate test of the truth of a sentence in LT is whether or not the sentence is a substitution instance of a known sentence. The corresponding ultimate test of the truth of most SIR1 sentences is whether or not certain links exist in the model. Every SIR1 sentence is a propositional function of link-predicates. A link-predicate is true of the model if it exists as an explicit link in the model, or if it can be deduced from axioms or higher-order link-predicates explicit in the model. Therefore, for the ultimate test of the truth of a link-predicate, TP must contain subprograms for eliminating ϵ-quantifiers. For example, $(\forall\alpha\epsilon x)[P[\alpha]]$ is true of the model if $P[\mu]$ is true of the model, for every object μ such that $\mu\epsilon x$ is true of the model. Thus, the ϵ-quantifier structure of SIR1 sentences serves as an important guide for the theorem-proving program.

c. The problem of implementing the "Exception Principle," discussed in Section A.3.c above for SIR, is still with us in SIR1. This means that the use of different sets of "truths" extracted from the model may lead to different answers to the same question. The solution to this problem is simply to be very careful in building and expanding the list of "truths" used by TP. I believe the iteration described in a. above is adequate, since it introduces the most closely related facts first. However, some experimentation in this area, once a working TP system is developed, will certainly be of interest.

In summary, an English question should be answered "yes" by the generalized semantic information retrieval system if and only if TP can prove the truth, with respect to the model, of the SIR1 sentence which corresponds to the question. TP attempts to prove the truth of sentences by going through the following steps:

i) Test whether the sentence is immediately implied by direct links in the model.

ii) Create a list of the axioms and link-predicates in the model which are most closely related to the sentence. Attempt to deduce the truth of the sentence from this list of truths, using both logical transformation methods such as those of LT, and model-dependent methods such as elimination of ϵ-quantifiers.

iii) After a reasonable amount of effort, add to the list of truths the axioms and link-predicates which are next-most-closely related to the sentence.

Repeat steps ii) and iii) until proof is completed or abandoned.

Note that TP operates in the finite domain of the propositional calculus. No provision has been make for true quantificational deductions, such as proving in general

$$(\exists y)(\forall x)P[x;y] \Rightarrow (\forall x)(\exists y)P[x;y]$$

Therefore TP could not, for example, perform the derivations of Appendix II which relate SIR and SIR1. The problem TP does attack is that of selecting relevant information from a large (although finite) store in order to construct proofs efficiently. Of course, a similar program for quantificational deduction would be a welcome addition to TP.

3) Complex question-answering: Some of the questions which SIR can answer require the system to perform more elaborate information retrieval tasks than simply testing the truth of an assertion. The answers to questions like, "How many fingers does John have?" and "Where is the book?" must be computed by searching and manipulating the data stored in the model in order to create appropriate responses.

Let us define a "question type" as a class of questions whose answers are found by following the same computational procedure. Questions of the same type generally differ from each other by referring to different objects in the model; those object-names are inputs to the computational procedure. In the previous sections we have considered the special type of all "yes-or-no" questions. In SIR, this class of questions was considered to be made up of many different question types -- one for each SIR relation -- and there was a corresponding multiplicity of computational procedures. In SIR1, the computational procedure for all "yes-or-no" questions is simply TP. However, TP requires as an input not just the names of objects, but rather the complete SIR1 sentence which corresponds to the question.

Unfortunately, no other SIR question types can be combined easily for a more general system. Each question type requires a different procedure for searching through the network of links, identifying useful information when it is found, and manipulating the information to produce the answer. Computer programming languages are well suited for specifying computational procedures, and for reasons described in Section III.A, the LISP language was quite convenient for specifying the complex question-answering procedures of SIR. However, as one attempts to enlarge and generalize SIR it becomes obvious that these programs should be made easier to write and easier to understand wherever possible. The full generality of LISP must be kept available, since new question types may require, in the answering process, unanticipated kinds of data manipulation; but the devices described below may be used to simplify the construction of question-answering programs.

In LISP, the flow of control within a program is normally determined by special functions called "predicates." The LISP system evaluates each predicate according to built-in or separately provided evaluation procedures, and chooses the next operation to performed according to whether the value of the predicate is "T" or "NIL" (corresponding to "true" or "false"). The SIR1 procedure-specification language should be similar to LISP, but should also allow the use of an additional class of predicates: namely, statements whose LISP values are "T" if a particular SIR1 sentence is true with respect to the model, and "NIL" otherwise. The procedure for evaluating these additional predicates would be just the procedure ordinarily used by SIR for determining the truth of SIR1 sentences, namely TP. Thus the full power of the SIR "yes-or-no" type of question-answering procedure could automatically be used within the procedure for answering a more complex type of question. Suppose that in the course of the procedure for answering the question, "What is the relative position of *x*?" it is determined that *y* is to the right of *x* and also that a *z* is to the right of *x*. The procedure could then contain the statement,

if $(\exists \alpha \epsilon z)[\mathrm{rightb}[\alpha;x] \wedge \mathrm{rightb}[y;\alpha]]$ *then* go A *else* go B

where A and B are locations of appropriate further instructions in the procedure. The procedure writer need not consider how to answer the question, "Is a *z* between *x* and *y*?" for TP will do that for him.

As a special application of this method for procedure-writing, let us consider how to obtain "no" or "sometimes" answers to questions of the "yes-or-no" type. The existence of separate programs for each relation in SIR permitted the consideration of special properties of the relation in determining an appropriate reply. In our generalized system, TP can reply "yes" if the SIR1 sentence *S* corresponding to the question is provable; otherwise the reply must be "insufficient information." Although a "no" answer cannot be obtained by TP directly, we *can* build into TP the ability to make a negative reply if it determines that the sentence ~S is provable; but no general change to TP can account for special properties of individual relations. However, this flexibility of SIR is recovered in the generalized system, without relinquishing any of the uniformity and generality of the SIR1 formalism and the TP program, by the use of simple procedures written in the LISP-plus-TP specification language. For example, the procedure for answering the question, "Is an *x* a *y*?" might be as follows:

if $(\forall \alpha\epsilon x)[\alpha\epsilon y]$ then YES;

else if $(\forall \alpha\epsilon x)[\sim\alpha\epsilon y]$ then NO;

else if $(\forall \alpha\epsilon y)[\alpha\epsilon x]$ then SOMETIMES;

else (INSUFFICIENT INFORMATION)

There remains the problem of implementing the specification language on a computer. When TP is available, it will be a simple matter to design an interpreter which would route control between TP and the LISP interpreter. Whether a compiler for these procedures is feasible depends on many factors, including the precise form of the TP system. The point here is that implementation of this procedure-specification language, a key part of the generalized semantic question-answerer, is feasible at the present state of the programming art.

In summary, a simple formalism has been presented which adds to LISP the truth-testing power of TP. This procedure-specification language, together with the SIR1 formalism, a corresponding word-association model structure, and the TP truth-testing program, constitute the basis for a "generalized" semantic information retrieval system. On the basis of information gleaned from the development of SIR, I have been able to describe this "generalized" system which has all the question-answering ability of SIR and accepts a much larger class of questions. More importantly, new relations can be added to the "generalized" system and the axioms of its proof procedure can be modified without any reprogramming, and question-answering procedures can be introduced and modified much more easily than they can be in SIR.

Chapter VII: Conclusions

A. Results.

1) Question-answering effectiveness: Chapter I described how question-answering behavior is a measure of a computer system's ability to "understand." SIR represents "meanings" in the form of a word-association, property-list model. As a result SIR is more general, more powerful, and, judging from its conversational ability, more "intelligent" than any other existing question-answering system. With respect to the fundamental problems of the other systems discussed in Chapter II:

a) SIR is not limited to a rigid prepared data structure and corresponding programs with specific, built-in, ad hoc definitions of "meanings" as is the "Baseball" program. Rather, it constructs its data structure as information is presented to it, and interprets "meanings" from "learned" word associations.

b) SIR is not restricted to the sentence-by-sentence matching of Phillips' "Question-Answering Routine." Instead, the SIR model provides access to relevant stored facts in a direct, natural way.

c) SIR, unlike SNYTHEX, does not require grammatical analyses which become more detailed and more complicated as the system expands. Instead, question-answering is based on semantic relationships, and the program structure can be simplified while enlarging the scope of the system in the manner described in Chapter VI.

d) The SIR model is not tailored for a single concept like the family relationships of SAD-SAM. However, the property-list structure of the model can easily be used to represent various special-purpose models and thus take advantage of their benefits, while permitting the storage of any relational information.

e) The SIR system is not restricted to testing the universal truth of a complete statement, regardless of the meanings of its components, as is Darlington's program. Rather, SIR procedures can be devised to answer any form of question, and the answers are based on SIR's current "knowledge" as determined by word associations in the model.

f) Although conceptually similar to Bennett's word relation system, SIR represents a vast improvement in that its list-structure model permits a direct representation for arbitrary word relations; the system contains programs for handling several different relations and their interactions; and both input formats and program logic may easily be modified.

2) Communication language: SIR provides a framework for reasonably natural communication between people and computers. Although somewhat stilted, both the input and the response languages used by SIR are sufficiently close to natural English to be easily understood by an untrained human. The input format recognition process used in SIR (Section IV, B) illustrates how far one may go toward "understanding" natural language, in the sense of recognizing word associations, without reference to grammatical structure. Of course, such a scheme cannot be generalized to cover any large portion of a natural language. It was used here simply as a device to get past the input phase and into the problems of representation and retrieval. However, this format matching process can easily be expanded to handle any sufficiently small portion of English.

Even in its present primitive state the process is not excessively restrictive to the untrained user. With the present system, the user could be instructed to present in complete English sentences simple facts and questions, and not to use any sentences with subordinate clauses, adjectives, conjunctions, or commas. These sentences may be about class relations, part-whole relations (possibly involving numbers), possessions, and left-to-right ordering relations. When used in a time-sharing environment (11) in which each sentence receives an immediate response, the system would have the effect of a "teaching machine" in training its user to restrict himself to recognizable sentence forms. After a few trial runs the programmer can easily add any new sentence forms which frequently arise, thus improving the chances of success for the next user. If this training process is too slow, the new user could study sample conversations from previous tests, or refer to an outline of available formats, before composing new statements to SIR. These processes are much simpler than learning a "programming" language. A sorted list of formats and more sophisticated similarity tests in the matching procedure would allow the addition of many more formats to the system with no corresponding increase in time required for recognition.

At the output end, the system demonstrates that "intelligent" responses are frequently possible without an elaborate generative grammar, as long as one can anticipate the classes of responses and frame each class in a suitable format.

3) <u>The model</u>: An important feature of SIR is the flexibility of the property-list structure of the model. Independent or related facts can automatically be added to or extracted from the system, and the same data may be expressed in more than one way.

Several existing computer systems, <u>e.g.</u> airline reservation systems, permit dynamic fact storage and retrieval. However, they depend upon the use of fixed, unique representations for the information involved. In SIR, there can be many representations which are equally effective in providing correct answers. <u>E.g.</u>, the system "knows" that the statement, "A finger is part of John" is true if (a) there is an explicit part-whole link from FINGER to JOHN; or if (b) there are links by means of which the retrieval programs can deduce that a finger is part of a person and John is a person; or if (c) there are links by means of which the retrieval programs can deduce that a finger is part of a hand, and a hand is part of John; _etc_. In addition, the system can automatically translate from one representation to another having some advantages. _E.g._, the "streamline" operation described in Section V.B, reduces storage space requirements by removing redundancy in the representation, without making any changes in the system.

The property-list model turns out to have advnatages even when another form of model seems more natural. For example, left-to-right spacial relations seem most easily represented by a linear ordering; _i.e._, "_x_ is to the left of _y_" could be modeled by placing _x_ ahead of _y_ in a left-to-right list. However, incomplete information can cause trouble for such a model. If it is known that "_x_ is to the left of _y_" and "_z_ is to the left of _y_," the linear ordering system cannot uniquely model the relative positions of _x_, _y_, and _z_. The property-list system, on the other hand, represents exactly the relations which are known; and the linear ordering of the objects can be _deduced_ from the property-list model, as is done in SIR by the "locate" function, if the data is sufficiently complete.

4) _Present state_: The processing time per statement for the SIR system with a standard LISP configuration on an IBM 7094 computer with 32K words of memory was about one second. All the examples prepared for Figure 1 and Figure 5 of this paper, including loading and compiling

all programs, took about 6 minutes of computer time. The SIR system, with all the relations, processing programs, and language formats described in this paper, utilizes almost the full capacity of the computer.

It must be remembered that the SIR system was not designed to solve any particular practical question-answering problem. It consists of a collection of relations which were introduced, as described in Section III.D, in order to investigate the various features and possibilities of the model. These relations do not necessarily bear any other useful or logical relationships to each other.

Although cramped for memory space, the present system has been successful in the sense that it has demonstrated the usefulness of the word assocaition property-list model, and it has suggested the more general system described in Chapter VI which extends the uses of the same model.

The scope of the present system indicates that it would be feasible to use the SIR model and present program organization in a practical information retrieval system for an IBM 7090 size computer, provided the system involved a reasonably small number of relations whose interactions are clearly understood. One possible application is a retrieval system which has been proposed at the RAND corporation for information about documents in Soviet cybernetics.(24) In that system the users will be interested in indirect relationships and implications, as well as the storage and retrieval of specific facts concerning authors and subjects of technical papers.

5) <u>Question-answering details</u>: The following points, although obvious in hindsight, did not become apparent until the program was fairly well developed:

a) A question-answering system cannot give definite negative replies without special information about the completeness and consistency of its data. The fact that SIR does not have such information accounts for frequent occurrences of the "INSUFFICIENT INFORMATION" response in places where a clearcut "NO" would be preferred.

b) If <u>x</u> stands in relation <u>R</u> to <u>y</u>, then a one-way link, <u>e.g.</u>, from <u>x</u> to <u>y</u> through attribute <u>R1</u> on the property list of <u>x</u>, may be sufficient for most question-answering applications. However, in the course of expanding the system the reverse link, from <u>y</u> to <u>x</u> through attribute <u>R2</u> on the <u>y</u> property-list, may be much more convenient. To allow for any eventuality in a general system both links should be provided from the start. Two-way links also provide the accessibility needed to experiment with various tree-searching procedures.

c) It is frequently possible for search procedures, even when unsuccessful, to provide extremely useful information to the user or programmer by specifying <u>why</u> they were unsuccessful. This point is discussed further in Section IV.C.

B. Extensions of SIR.

1) <u>Adding relations</u>: Two major obstacles, in addition to computer memory size, stand in the way of extending a SIR-like system by adding new relations and their associated programs: (a) the problem of interaction between a new relation and those already in the system, requiring modifications throughout the system for even minor additions; and (b) the problem of the time required to search through trees of words linked by relations. This time apparently must grow exponentially as the number of relations increases.

The problem of interactions can best be overcome by replacing SIR with a generalized system. As discussed in Chapter VI, this change would greatly reduce the interaction problem and simplify the introduction of new relations. In addition, the programs would probably be significantly smaller in the generalized system. Not only would all "yes-or-no" type question-answering programs be replaced by a single, "theorem-proving" program; in addition, the procedure specification language of the generalized system would result in more compact, as well as more readable, programs.

The other obstacle to the expansion of a semantic information retrieval system is the same obstacle which occurs in programs for theorem proving, game playing, and other areas of articicifcal intelligence -- the problem of searching through an exponentially growing space of possible solutions. Here there is no basic transformation that can be made to avoid the mathematical fact that the number of possible interconnections between elements is an exponential function of the number of elements involved. This means that in SIR, the time required to search for certain relational links increases very rapidly with both the number of individual elements which can be linked and the number of different relations which can do the linking. However, many of the heuristics for reducing search effort which have been suggested in other areas concerned with tree-structured data can be applied here.

In the first place, relations seem to be divided into independent (non-interacting) groups; e.g., spatial relations are quite independent of temporal relations. The search space affected by a new relation is really just the space of interacting relations, which may be a very small subset of the total space of relations. The axioms of the generalized system can be used to identify the groups of interacting relations. Secondly, the existence of two-way links permits the search for a path between two points in the data structure to proceed from either end (whichever is likely to produce a more efficient search), or possibly from both ends simultaneously toward an unknown common point. Finally, semantic information in the model might be useful in suggesting intermediate points to use as "stepping stones" in a larger tree search, thus greatly reducing the search effort. I believe that the use of these and similar heuristic devices, along with expected increases in computer speed and memory size and the introduction of parallel processing computer hardware, will make a large-scale semantic information retrieval system practical.

2) <u>Adjectives and n-ary relations</u>: All the relations in the present system are binary relations. The model can be extended to handle arbitrary n-ary relations as follows:

a. Unary operators could be simply flags on the property lists of the objects to which they apply. Or, if for purposes of uniformity we forbid the use of flags, then they could be attributes whose values are always a dummy symbol which indicates that the attribute is to be interpreted as a unary operator. In handling adjectives, the following decision would have to be made: should an adjective be modeled by an unary operator, or should it be the value of some attribute? For example, "little red schoolhouse" could be represented in the model in any of the following ways:

i) An object which is an element of the set "SCHOOLHOUSE," and which has on its property list the flags "LITTLE" and "RED."

ii) The same object, which has on its property list the attribute "MODIFIERS" with associated value "(LITTLE, RED)."

iii) The same object, which has on its property list the attribute-value pairs "(SIZE, LITTLE)" and "(COLOR, RED)."

The second representation is equivalent to the first but avoids the need for unary operators. The third representation contains the most information and is most consistent with the present form of the SIR model, but has the disadvantage that it requires the use of a dictionary to establish appropriate classifications of adjectives. The "best" representation to use would have to be determined by experimentation and would depend upon the organization of the information retrieval programs which use the model.

b. Trinary (*e.g.*, those involving transitive verbs) and higher order relations could be represented in various ways analogous to the treatment of binary relations. *E.g.*, the n-ary relation *R* can be factored into *n* relations *R1*, *R2*, ..., *Rn*, such that

$$\langle x_1, x_2, \ldots, x_n \rangle \in R \text{ if and only if}$$

$$\langle x_2, \ldots, x_n \rangle = R1[x_1] \wedge \langle x_1, x_3, \ldots, x_n \rangle = R2[x_2] \wedge$$

$$\ldots \wedge \langle x_1, x_2, \ldots, x_{n-1} \rangle = Rn[x_n],$$

where the value of the attribute *Rj* on the property list of *xj* would be the ordered sequence $\langle x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n \rangle$. More specifically, the trinary relation established by the statement, "John gave a book to Jim" could be factored into the three relations "GIVER," "GIVEN," and "GETTER." The propety list of "JOHN" would have the pair "(GIVER, (BOOK, JIM))," the property list describing "BOOK" would contain "GIVEN, (JOHN, JIM))," and "(GETTER, (JOHN, BOOK))" would be placed on "JIM's" property list. Once again, the practicality and efficiency of such a representation can only be discovered by developing and experimenting with working computer programs.

3) Next steps: The present SIR system, and its generalized version discussed in Chapter VI, are only first steps toward a true "understanding" machine. Eventually we must solve the "advice-taker" problem (22), which involves controling the operation of the machine merely by "advising" it, in a suitable English-like language, of the desired procedures or results.

One approach to the "advice-taker" is to develop programs which can produce other programs in accordance with simple instructions. Such program writing programs could be an outgrowth of current work on computer language "compilers," if the input and output forms are sufficiently well-defined. Simon (39) is working on this approach by developing a system which accepts a broad range of English statements as input to such a program-writing program.

SIR suggests an alternative approach. Rather than developing a program which writes other programs to do specified tasks, I propose we develop a single, general program which can do any task provided the program is properly controlled by information in its model. "Giving advice" would then require only the relatively simple process of inserting appropriate control information into the model. The SIR model provides its programs with information about the truth of particular relations between specific objects. The model in the generalized system also provides the "theorem-prover" program with axioms which describe properties of relations and interactions between relations. The next generalization should involve adding to the model information which will specify and control theorem-proving and model-searching procedures for the program.

After the above two approaches to an "understanding" machine have been developed independently, they should be synthesized. The program-writing program should be incorporated into the general program of the model-dependent system. The resulting system would then be able to construct arbitrary procedure specifications, in accordance with simple instructions which had been placed in its model.

Ultimately the "intelligent" machine will have to be able to abstract from the information in its model, "realize" the necessity for additional action, and create the necessary instructions for itself. The design of such an "artificial intelligence" awaits the development of automatic concept formation and inductive inference systems (20,41) as well as the generalizations of SIR described above.

C. Concerning Programming.

1) Value of programming: Many of the results and conclusions written after the development of a large computer program such as SIR frequently appear as if they could have been established without the tedious effort of programming. This is rarely true, and in fact, new systems which are described as complete "except for the programming" usually require fundamental modifications if and when they are translated into operating programs. The reasons for the importance of actually writing the program include the following:

a) Without a program it is extremely difficult to tell whether the specifications for a system are really complete and consistent. Crucial decisions may be considered minor details, and contradictions may go unnoticed, until one is compelled to build an operating system.

b) The process of programming not only turns up fallacies in the specifications for a system, but also generally suggests ways for avoiding them and improving the system. Thus programming can be much more valuable than just searching for errors in the original specification. A completed "debugged" programmed system usually turns out to be a compromise between the system as it was originally specified, a simpler system which was more feasible to actually construct, and a more elaborate system whose new features were thought of during the programming process. This resulting system is frequently as useful and certainly more reliable than the originally specified system, and in addition it may suggest the design of even more advanced systems. With SIR, for example, methods for implementing the "exception principle" and resolution of ambiguities arose from the design of the basic question-answerer, and the specifications for the generalized system of Chapter VI are based largely on properties of the final, working SIR system.

c) The programming process frequently turns up insights which might not otherwise be discovered (see for example paragraph A.5 above).

d) Finally, the resulting program provides at the same time a demonstration of the feasibility of the ideas upon which it is based, a measure of the practicality of the system in terms of time and space requirements, and an experimental device for testing variations in the original specifications.

2) Uniformity of representation: A uniform tree linkage and search procedure would simplify coding and allow the programmer to concentrate on the more important problems of program organization and search strategies. Such a standard representation would have to be flexible enough to handle the most complicated cases. In SIR, the uniform use of only type-3 links on all property-lists and only type-1 links on all sub-property-lists would probably achieve the desired result. An alternative, somewhat more complicated (but more economical of storage) way to achieve the same result of freeing the programmer from concern for details, would be to allow several kinds of linkages to be used wherever they were best suited (e.g., type-1, -2, and -3 links), but require all retrieval programs to be able to recognize the type of a link and treat each one appropriately.

If this alternative of allowing the use of several types of linkages were used in the generalized system, the nature of the links appropriate for particular relations could be stored in the model on the property-lists of the relations. In this way the type-identification would be readily available to the retrieval programs.

3) __Programming tree-search__: In order to handle some of the retrieval processes I had to develop some general tree-tracing functions. The facility in the LISP language for defining functions of functional arguments permitted the design of programs providing a powerful ability to specify complex search procedures. For example, one of the most useful functions was "find[start; link; test]," where "start" can be any word in the model structure, "link" specifies which attribute to use to find succeeding words, and "test" is the name of a function to be applied in turn to each word reachable from "start" along the kind of path specified by "link." If the value of "test" applied to a word is the special symbol "NIL," the search continues; otherwise the value of "find" (and the result of the search) is just the value of "test." This result may contain the word which satisfied the test and the successful path, _i.e._, the list of words which link "start" to the selected word in the desired way. Note that the function "find" can be cascaded, _i.e._, "test" can be another application of "find" itself. _E.g._, in testing whether every A is part of some B, we may wish to test whether there is a class _u_ such that every A is a _u_ and every _u_ is part of some B. This test is carried out simply by executing the following function (given in LISP meta-language notation), and testing whether its value is "NIL" or not:

find[A; SUPERSET; λ[[u][find [u; SUPERPART-OF-EACH; λ[[v][v=B]]];B]]]].

If a uniform representation (as described in paragraph 2, above) had been used throughout SIR, then it would have been easy to develop a complete set of general network-tracing functions like "find." Such a set of functions could be the basis for a language which makes programming tree- and network-searching systems much simpler than it is now. Such a language might thus contribute to research in the areas of pattern recognition, game-playing (36), and network analysis as well as semantics and information retrieval. Note that the success or failure of an application of the function "find" depends only on the connectivity of the network; the order in which nodes are generated and tested, and therefore the efficiency of the system for various kinds of networks, must be decided in advance and built into the definition of the function.

4) Program simplification: The "procedures" presented in section V.A which were described as "rough flow charts" for the retrieval programs, may seem unnecessarily complicated. This is true for the following reasons:

a) Each procedure was written as an explanation of how a particular program operates, and the place of these programs in the over-all program structure was de-emphasized to avoid confusion. There is must more hierarchical structure and use of common subroutines in the actual SIR program than is indicated in those procedures.

b) As with most programming tasks, many possible simplifications occur to the programmer as after thoughts. If I started over now, I could certainly construct a neater, more compact SIR system -- especially by incorporating some of the ideas discussed in paragraphs 2 and 3 above. However, I would be more inclined to ignore SIR altogether and instead start programming the generalized system of Chapter VI.

c) Unfortunately, many of the "simple" reasoning procedures the program must go through really are complicated. It was surprising to me how many possible routes one may take to deduce a simple fact like, "A is part of B."

D. Subjects for Future Experiments.

1) <u>Search procedures</u>: The relative merits of different tree-searching procedures should be investigated, since any device which significantly reduced search effort would be a valuable contribution to the practicality of SIR-like systems. In seeking a path between two nodes, for example, one might compare the procedure of moving one ply from each end, alternately, and looking for a common node, with the procedure of continually branching out from one node, searching for the other. Even this latter procedure can be performed in either a "breadth first" or a more naturally recursive "depth first" manner. While the first procedure mentioned above cuts the effective depth of a successful search in half, it also introduces matching problems in order to recognize success, and makes it more difficult to discover the complete successful path. Which of the various procedures is "best" will depend on the size of the networks, the relative frequency of success, the average length of successful paths, etc. Therefore the best way to determine the most efficient methods is to experiment on an operating system, preferably with respect to a particular problem area.

2) <u>Linkage structure</u>: The optimum number of explicit links needed should be investigated. One might expect a trade-off here between space and time; <u>i.e.</u>, that a removal of redundant links, for instance by "streamlining" operations, should save storage at the expense of increasing the average question-answering time, while introducing redundant links, for instance by adding as explicit links all question-answers which are successfully obtained, should use up space but speed up the question-answering process. However, this trade off is not strictly necessary.

Explicit links save time only when they provide correct answers; otherwise they use time by requiring spurious parts of the network to be searched. Which redundant links to weed out, as well as which search procedure to use, depends on the characterisitics of the model and questions in a particular application and must be determined by experimentation.

Another structuring problem to be considered is that of consistency. At present SIR tries to test the consistency of each input sentence with the information it already has stored, before adding the new relations to the model. It might be more efficient to blindly accept each input sentence independently, and then check the consistency of the model from time to time, say between input sentences, "complaining" if problems occur. This procedure would give later information equal precedence with earlier inputs, which might be a preferred arrangement for some applications.

3) Ambiguity in language: A system similar to SIR could be used as a basis for a study of ambiguity in language. The example given above in section V.B shows how SIR can resolve an ambiguous word meaning on the basis of related word meanings. Similarly an expanded version of SIR might be able to resolve ambiguous sentence structure on the basis of the meanings (or, more precisely, the contents of the property-lists) of the words in the sentence. Thus the system could be as effective as people in recognizing the structural difference between sentences like,

"Bring me the bottle of milk which is sour," and

"Bring me the bottle of milk which is cracked."

Such a study might contribute to our knowledge of the use of language and how people resolve ambiguities. It could investigate how much semantic or contextual information is needed to resolve ambiguities which give people trouble, such as "They are flying planes."

4) Simulation: The behavior of SIR in answering questions and resolving ambiguities suggests that the program "understands the meanings" of the words in its model. The information SIR associates with a word by means of the property-list of the word is analagous to the information a person associates with an object by means of a "mental image" of the object. Perhaps we can carry this analogy further and say that since certain aspects of the behavior of SIR are similar to human behavior, then the representation and manipulation of data within SIR is similar, at the information processing level, to the representation and manipulation procedures a person carries out when "thinking."

Psychologists have simulated on a computer human problem-solving behavior (28) and the process of memorizing nonsense syllables (14). Perhaps SIR can be considered a simulation of the human process of learning and thinking about coherent facts. Psychological experiments would have to be devised to test this theory by testing more precisely the similarity of SIR's behavior to human behavior. In the process we might obtain valuable ideas for both improving the model and understanding human cognitive processes.

BIBLIOGRAPHY

1. ACF Industries, Avion Div. "Translating From Ordinary Discourse Into Formal Logic -- A Preliminary Study," Scientific Report AF CRC-TN-56-770.

2. Bennett, J. L. "A Computer Program for Word Relations," Memo 1961-1, Mechanical Translation Group, RLE, MIT. Cambridge, Mass. 1961.

3. Bobrow, D. G. "Syntactic Analysis of English by Computer -- A Survey," Proc. FJCC. Spartan Press, 1963.

4. Bobrow, D. G., and Raphael, B. "A Comparison of List-Processing Computer Languages," Comm. ACM, May or June 1964.

5. Carnap, R. Meaning and Necessity. U. of Chicago Press, Chicago, Illinois. 1947.

6. Carnap, R. "Foundations of Logic and Mathematics," International Encyclopedia of Unified Science, Volume 1, no. 3. U. of Chicago Press, Chicago, Illinois. 1939.

7. Carroll, J. D., Abelson, R. P., and Reinfeld, W. "A Computer Program Which Assesses the Credibility of Assertions," draft. Yale University. July 1963.

8. Charney, E. "Word-meaning and Sentence-meaning," abstract in Mechanical Translation, Volume 7, no. 2. 1963.

9. Chomsky, N. Syntactic Structures, Mouton and Co. 1947.

10. Cohen, D. "Picture Processing in a Picture Language Machine," National Bureau of Standards Report 7885. April 1962.

11. Corbato, F. J., et. al. The Compatible Time-Sharing System. MIT Press, Cambridge, Mass. 1963.

12. Darlington, J. L. "Translating Ordinary Language into Symbolic Logic," abstract in Mechanical Translation, Volume 7, no. 2. 1963.

13. Davis, M., and Putnam, H. "A Computational Proof Procedure," AFOSR TR 59-124. Rensselaer Polytechnic Institute, Troy, N.Y. 1959.

14. Feigenbaum, E. "The Simulation of Verbal Learning Behavior," Proc. WJCC, Volume 19. 1961.

15. Freudenthal, H. LINCOS: Design of a Language for Cosmic Intercourse. North Holland Press, 1960.

16. Fries. The Structure of English. Harcourt, Brace, New York, 1952.

17. Green, B. F., Jr., et. al. "Baseball: An Automatic Question-Answerer," Proc. WJCC, Volume 19. 1961.

18. Kazemier, B. H., and Vuysje, D., eds. The Concept and the Role of the Model in Mathematics and Natural and Social Sciences. Gordon and Breach Science Publishers, N.Y. 1963.

19. Klein, S. "Some Experiments Performed with an Automatic Paraphraser," abstract in Mechanical Translation, Volume 7, no. 2. 1963.

20. Kochen, M. "Experimental Study of 'Hypothesis Formation' by Computer," Proc. 4th London Symposium on Information Theory. C. Cherry, ed. London, 1961.

21. Lindsay, R. K. " A Program for Parsing Sentences and Making Inferences about Kinship Relations," Proc. Western Management Science Conference on Simulation. A. Hoggatt, ed. to be published.

22. McCarthy, J. "Programs with Common Sense," Proc. Symposium on Mechanization of Thought Processes. National Physics Laboratory, Teddington, England. Her Majesty's Stationery Office, London. 1959.

23. McCarthy, J., et. al. LISP 1.5 Programmer's Manual. MIT Press, Cambridge, Mass. 1963.

24. Maron, I. RAND Corp., Santa Monica, California. private communication, 1963.

25. Minsky, M. "Steps Toward Artificial Intelligence," Proc. IRE, special computer issue, 1961.

26. Newell, A., ed. Information Processing Language V Manual. Prentice Hall, Englewood Cliffs, N.J. 1961.

27. Newell, A., et. al. "Empirical Explorations of the Logic Theory Machine: A Case Study in Heuristics," Proc. WJCC, IRE, 1957.

28. Newell, A., et. al. "Report on a General Problem-Solving Program," Proc. International Conference on Information Processing. Paris, UNESCO House, 1959.

29. O'Donnell, M. The New Day In and Day Out. Row, Peterson and Co. Evanston, Illinois. 1948.

30. Ogden, C. K. Basic English. Paul, Trench, Trubner and Co. London. 1932.

31. Phillips, A. V. "A Question-Answering Routine," MIT Mathematics Dept. Master's Thesis. Cambridge, Mass. 1960.

32. Quillian, R. "A Revised Design for an Understanding Machine," Mechanical Translation, Volume 7, no. 1, 1962.

33. Quine, W. Word and Object. MIT Press, Cambridge, Mass. 1960.

34. Reichenbach, H. Elements of Symbolic Logic. The Macmillan Co., N.Y., 1947.

35. Research Lab. of Electronics and Computation Center, MIT, COMIT Programmer's Reference Manual. MIT Press, Cambridge, Mass. 1961.

36. Samuel, A. L. "Some Studies in Machine Learning Using the Game of Checkers," IBM J. of Research and Development, Volume 3, no. 3. 1959.

37. Shaw, C. J. "JOVIAL and Its Documentation," Comm. ACM, Volume 3, no. 6, 1963.

38. Simmons, R. F., et. al. "Toward the Synthesis of Human Language Behavior," SP-466, Systems Development Corp., Santa Monica, Cal.

39. Simon, H. A. "Experiments with a heuristic compiler," Paper P-2349, RAND Corp., Santa Monica, California. 1961.

40. Slagle, J. "A Computer Program for Solving Problems in Freshman Calculus," J. ACM, Jan., 1964 and Doctoral Dissertation, Mathematics Department, MIT. May 1961.

41. Solomonoff, R. J. "An Inductive Inference Machine," IRE National Convention Record, pt. 2, pp. 56-62, 1957.

42. Sommers, F. T. "Semantic Structures and Automatic Clarification of Linguistic Ambiguity," International Electric Corp., Paramus, N.J., 1961.

43. Suppes, P. Introduction to Logic. Van Nostrand Co., Princeton, N.J. 1957.

44. Ullman, S. Words and Their Use. Philosophical Library, N.Y. 1951.

45. Walpole, H. R. Semantics: The Nature of Words and Their Meanings. W.W. Norton and Co., N.Y. 1941.

46. Wang, H. "Toward Mechanical Mathematics," IBM J. of Research and Development, Volume 4, no. 1. 1960.

47. Ziff, P. Semantic Analysis. Cornell U. Press, Ithaca, N.Y. 1960.

Appendix I: Notation

A. Basic Symbols.

The purpose of this section is to present some of the formal logical terminology used in this paper. In the following list, the use of various symbols will be explained by means of definitions, examples, or statements of interpretation.

Symbol	Explanation
...	and so forth.
A,B,C, ...	meta-symbols standing for any logical formulas.
$\sim \wedge \vee \Rightarrow \Leftrightarrow$	the propositional connectives.
$\sim A$	not A; A is false.
$A \wedge B$	A and B (are both true).
$A \vee B$	A or B or both.
$A \Rightarrow B$	A implies B.
$A \Leftrightarrow B$	A if and only if B.
x,y,z, ...	variables; names of unknown objects or sets.
α, β, γ, ...	constants; names of particular objects or sets.
$\alpha \in x$	α is a member of the set x.
$x \subseteq y$	set x is contained in set y.
$\alpha \notin x$	[α x]; α is not a member of the set x.
x=y	x and y are the same object or set.
$x \neq y$	[x=y].
$\forall$	universal quantifier symbol.
$(\forall x)$	universal quantifier.
$(\forall x)A$	A is true for all values of x.
$\exists$	existential quantifier symbol.
$(\exists x)$	existential quantifier.
$(\exists x)A$	there exists an x such that A is true.
$\{\alpha, \beta, \gamma, \ldots\}$	an unordered set of the objects named.
$\langle \alpha, \beta \rangle$	the ordered pair of the objects named.
=df	equals by definition; is defined to be.

B. Subordinate Proof Derivation.

"Subordinate proof" is a method for proving logical deductions in the first-order predicate calculus ("the quantificational calculus"). The formulation outlined here is due to Prof. Hartley Rogers, Jr. It is similar to the system of "general inference" described by Suppes (43).

Definition: Subordinate Proof Derivation of a formula $\underline{B}$ from a finite, possibly empty, set of formulas $\mathcal{Q}$ =df
an arrangement of formulas and long brackets satisfying the conditions:

1) The first $\underline{k}$ lines of the derivation consist of the formulas of $\mathcal{Q}$.

2) Given $\underline{n}$ lines of the derivation, the n+1st line may consist of any formula whatever, if a new long bracket is begun to the left of that formula inside all existing brackets not previously terminated.

Definition: In a Subordinate Proof Derivation, line $\underline{j}$ is called an <u>ancestor</u> of line ℓ if $j < \ell$ and line $\underline{j}$ occurs inside no long brackets other than those containing line ℓ.

3) Given $\underline{n}$ lines of a derivation, the n+1st line may consist of a formula $\underline{A}$ (without a new long bracket) if
 i) $\underline{A}$ is a known true theorem.
 ii) $\underline{A}$ is implied, in the propositional calculus, by any set of formulas in ancestor lines to the n+1st line, or
 iii) $\underline{A}$ can be obtained from a formula in an ancestor line by an allowable use of the method of US, UG, ES, EG, I1, or I2.

Definitions: Let $\underline{A}$ be any formula, and let $\underline{\alpha}$ and $\underline{\beta}$ be terms.
A^{α}_{β} =df the formula obtained from $\underline{A}$ by substituting $\underline{\beta}$ for every <u>free</u> occurrence of $\underline{\alpha}$ in $\underline{A}$, *i.e.*, for every occurrence of α not within the scope of a quantifier containing $\underline{\alpha}$.
 US =df Universal Specification, by which $(\forall\alpha)A$ becomes A^{α}_{β}.
 UG =df Universal Generalization, by which A becomes $(\forall\alpha)A^{\beta}_{\alpha}$.
 ES =df Existential Specification, by which $(\exists\alpha)A$ becomes A^{α}_{β}.
 EG =df Existential Generalization, by which A becomes $(\exists\beta)A^{\alpha}_{\beta}$.
 I1 =df A rule which allows insertion of a formula of the form $\alpha=\alpha$.
 I2 =df A rule by which $\{\alpha=\beta, A\}$ leads to A^{α}_{β}.
Certain conditions restrict the allowable usage of most of these quantifier transformation methods. These conditions, which relate to conflicts between variable interpretations and dependencies between constants, are too involved to present in this outline.

4) An innermost long bracket may be terminated at (and including) the $\underline{n}^{th}$ line if we write as the n+1st line $[A \Rightarrow C]$ where A and C are, respectively, the first and last formulas in the long bracket in question.

5) An innermost long bracket may be terminated at the $\underline{n}^{th}$ line if that bracket begins with a formula $\sim A$ and has for its last two lines C and $\sim C$, for some formula C, if we write A as the n+1st line.

6) The last line has no long brackets and is the formula B.

<u>Main Theorem (given here without proof)</u>: If there is a Subordinate proof Derivation of B from $\mathcal{Q}$, then B is quantificationally deducible from $\mathcal{Q}$.

Appendix II: Derivations of SIR Deduction Procedures

Each of the 23 deduction procedures listed in Table b. is a theorem of the SIR1 formal system. The proofs, presented below, generally consist of four statements:

i) The SIR deduction procedure, as stated in Table b.

ii) A corresponding SIR1 wff, obtained through use of the correspondences of Table c.

iii) The quantificational calculus statement obtained from the formula in ii) by eliminating ϵ-quantifiers as described in Section VI.B.

iv) The outline of a Subordinate Proof Derivation for the statement in iii). These proofs are "outlines" in the sense that occasionally several steps are combined into one, line numbers are used as meta-symbols to stand for lengthy expressions, and derived rules of inference such as "modes ponens" are used when convenient. However, enough detail and explanation is presented so that complete formal "SPD's" can easily be constructed if desired.

The axioms of SIR1, as given in Table d. and its associated definitions, are introduced into the Subordinate Proofs as "true" theorems whenever necessary. Universal quantification over all free variables in the initial and final statements in the following proofs is assumed.

In some cases, the proofs of SIR deduction procedures follow immediately from SIR1 axioms or definitions, so that "SPD's" are unnecessary.

1) $\mathcal{J}(\subset)$

$x \subset y \wedge y \subset z \Rightarrow x \subset z$

$(\forall\alpha)[\alpha\epsilon x \Rightarrow \alpha\epsilon y] \wedge (\forall\alpha)[\alpha\epsilon y \Rightarrow \alpha\epsilon z] \Rightarrow (\forall\alpha)[\alpha\epsilon x \Rightarrow \alpha\epsilon z]$

1. $(\forall\alpha)[\alpha\in x \Rightarrow \alpha\in y] \wedge (\forall\alpha)[\alpha\in y \Rightarrow \alpha\in z]$
2. $\beta\in x \Rightarrow \beta\in y$ — US1 (by US in line 1)
3. $\beta\in y \Rightarrow \beta\in z$ — US1
4. $\beta\in x$
5. $\beta\in y$ — 4,2
6. $\beta\in z$ — 5,3
7. $\beta\in x \Rightarrow \beta\in z$
8. $(\forall\alpha)[\alpha\in x \Rightarrow \alpha\in z]$ — UG7

$1. \Rightarrow 8.$ qed.

2) $x=y \Rightarrow x\subset y$

$x=y \Rightarrow (\forall\alpha\in x)[\alpha\in y]$

$x=y \Rightarrow (\forall\alpha)[\alpha\in x \Rightarrow \alpha\in y]$

1. $x=y$
2. $\sim(\forall\alpha)[\alpha\in x \Rightarrow \alpha\in y]$
3. $(\exists\alpha)\sim[\alpha\in x \Rightarrow \alpha\in y]$ — 2
4. $\sim[\beta\in x \Rightarrow \beta\in y]$ — ES3
5. $\beta\in x \wedge \sim\beta\in y$
6. $\beta\in y$ — I2-1,5
7. $\sim\beta\in y$ — 5
8. $(\forall\alpha)[\alpha\in x \Rightarrow \alpha\in y]$

$1. \Rightarrow 8.$ qed.

3) $\text{equiv}[x;y] \Rightarrow x\subset y$

$x=y \Rightarrow (\forall\alpha\in x)[\alpha\in y]$ same as 2).

4) $\alpha\in x \wedge x\subset y \Rightarrow \alpha\in y$

$\alpha\in x \wedge (\forall\beta\in x)[\beta\in y] \Rightarrow \alpha\in y$

$\alpha\in x \wedge (\forall\beta)[\beta\in x \Rightarrow \beta\in y] \Rightarrow \alpha\in y$

1. $\alpha\in x \wedge (\forall\beta)[\beta\in x \Rightarrow \beta\in y]$
2. $\alpha\in x \Rightarrow \alpha\in y$ — US1
3. $\alpha\in y$ — 1,3

$1. \Rightarrow 3.$ qed.

5) $\mathcal{T}(\text{equiv})$

$\mathcal{T}(=)$ axiom.

6) $\mathcal{R}$(equiv)

$\mathcal{R}(=)$ axiom.

7) $\mathcal{L}$(equiv)

$\mathcal{L}(=)$ axiom.

8) $\sim \mathrm{owng}[x;x]$

$\sim(\forall\alpha\in x)(\exists\beta\in x)[\mathrm{ownb}[\alpha;\beta]]$

$\mathcal{R}(\mathrm{ownb})$ axiom.

9) $\mathrm{owng}[x;y]\wedge z\subset y\Rightarrow \mathrm{owng}[x;z]$

$(\forall\beta\in y)(\exists\alpha\in x)[\mathrm{ownb}[\alpha;\beta]]\wedge(\forall\alpha\in z)[\alpha\in y]\Rightarrow(\forall\beta\in z)(\exists\alpha\in x)[\mathrm{ownb}[\alpha;\beta]]$

$(\forall\beta)[\beta\in y\Rightarrow(\exists\alpha)[\alpha\in x\wedge \mathrm{ownb}[\alpha;\beta]]]\wedge(\forall\alpha)[\alpha\in z\Rightarrow\alpha\in y]$
$\Rightarrow(\forall\beta)[\beta\in z\Rightarrow(\exists\alpha)[\alpha\in x\wedge \mathrm{ownb}[\alpha;\beta]]]$

1. $(\forall\beta)[\beta\in y\Rightarrow(\exists\alpha)[\alpha\in x\wedge \mathrm{ownb}[\alpha;\beta]]]\wedge(\forall\alpha)[\alpha\in z\Rightarrow\alpha\in y]$
2. $\gamma\in y\Rightarrow(\exists\alpha)[\alpha\in x\wedge \mathrm{ownb}[\alpha;\gamma]]$ US1
3. $\gamma\in z\Rightarrow\gamma\in y$ US1
4. $\gamma\in z\Rightarrow(\exists\alpha)[\alpha\in x\wedge \mathrm{ownb}[\alpha;\gamma]]$ 3,2
5. $(\forall\beta)[\beta\in z\Rightarrow(\exists\alpha)[\alpha\in x\wedge \mathrm{ownb}[\alpha;\beta]]]$ UG4

1.⇒5. qed.

10) $\mathrm{owng}[x;y]\wedge x\subset z\Rightarrow \mathrm{owng}[z;y]$

$(\forall\beta\in y)(\exists\alpha\in x)[\mathrm{ownb}[\alpha;\beta]]\wedge(\forall\alpha\in x)[\alpha\in z]\Rightarrow(\forall\beta\in y)(\exists\alpha\in z)[\mathrm{ownb}[\alpha;\beta]]$

$(\forall\beta)[\beta\in y\Rightarrow(\exists\alpha)[\alpha\in x\wedge \mathrm{ownb}[\alpha;\beta]]]\wedge(\forall\alpha)[\alpha\in x\Rightarrow\alpha\in z]$
$\Rightarrow(\forall\beta)[\beta\in y\Rightarrow(\exists\alpha)[\alpha\in z\wedge \mathrm{ownb}[\alpha;\beta]]]$

1. $(\forall\beta)[\beta\in y\Rightarrow(\exists\alpha)[\alpha\in x\wedge \mathrm{ownb}[\alpha;\beta]]]\wedge(\forall\alpha)[\alpha\in x\Rightarrow\alpha\in z]$
2. $\gamma\in y\Rightarrow(\exists\alpha)[\alpha\in x\wedge \mathrm{ownb}[\alpha;\gamma]]$ US1
3. $\gamma\in y$
4. $(\exists\alpha)[\alpha\in x\wedge \mathrm{ownb}[\alpha;\gamma]]$ 3,2
5. $\mu\in x\wedge \mathrm{ownb}[\mu;\gamma]$ ES4
6. $\mu\in x\Rightarrow\mu\in z$ US1
7. $\mu\in z\wedge \mathrm{ownb}[\mu;\gamma]$ 5,6
8. $(\exists\alpha)[\alpha\in z\wedge \mathrm{ownb}[\alpha;\gamma]]$ EG7
9. 3.⇒8.
10. $(\forall\beta)[\beta\in y\Rightarrow(\exists\alpha)[\alpha\in x\wedge \mathrm{ownb}[\alpha;\beta]]]$ UG9

1.⇒10. qed.

11) $own[x;y] \wedge x \subset z \Rightarrow own[z;y]$

$(\exists\alpha\epsilon x)[ownb[\alpha;y]] \wedge (\forall\alpha\epsilon x)[\alpha\epsilon z] \Rightarrow (\exists\alpha\epsilon z)[ownb[\alpha;y]]$

$(\exists\alpha)[\alpha\epsilon x \wedge ownb[\alpha;y]] \wedge (\forall\alpha)[\alpha\epsilon x \Rightarrow \alpha\epsilon z] \Rightarrow (\exists\alpha)[\alpha\epsilon z \wedge ownb[\alpha;y]]$

1. $(\exists\alpha)[\alpha\epsilon x \wedge ownb[\alpha;y]] \wedge (\forall\alpha)[\alpha\epsilon x \Rightarrow \alpha\epsilon z]$
2. $\beta\epsilon x \wedge ownb[\beta;y]$ ES1
3. $\beta\epsilon x \Rightarrow \beta\epsilon z$ US1
4. $\beta\epsilon z \wedge ownb[\beta;y]$ 2,3
5. $(\exists\alpha)[\alpha\epsilon z \wedge ownb[\alpha;y]]$ EG4

1.⇒5. qed.

12) $owng[x;y] \wedge z\epsilon y \Rightarrow own[x;z]$

$(\forall\beta\epsilon y)(\exists\alpha\epsilon x)[ownb[\alpha;\beta]] \wedge z\epsilon y \Rightarrow (\exists\alpha\epsilon x)[ownb[\alpha;z]]$

$(\forall\beta)[\beta\epsilon y \Rightarrow (\exists\alpha)[\alpha\epsilon x \wedge ownb[\alpha;\beta]]] \wedge z\epsilon y \Rightarrow (\exists\alpha)[\alpha\epsilon x \wedge ownb[\alpha;z]]$

1. $(\forall\beta)[\beta\epsilon y \Rightarrow (\exists\alpha)[\alpha\epsilon x \wedge ownb[\alpha;\beta]]] \wedge z\epsilon y$
2. $z\epsilon y \Rightarrow (\exists\alpha)[\alpha\epsilon x \wedge ownb[\alpha;z]]$ US1
3. $(\exists\alpha)[\alpha\epsilon x \wedge ownb[\alpha;z]]$ 1,2

1.⇒3. qed.

13) $\sim partg[x;x]$

$\sim(\forall\alpha\epsilon x)(\exists\beta\epsilon x)[partb[\alpha;\beta]]$

$\mathcal{R}(partb)$ axiom.

14) $partg[x:y] \wedge z \subset y \Rightarrow partg[x;z]$

$(\forall\beta\epsilon y)(\exists\alpha\epsilon x)[partb[\alpha;\beta]] \wedge (\forall\alpha\epsilon z)[\alpha\epsilon y] \Rightarrow (\forall\beta\epsilon z)(\exists\alpha\epsilon x)[partb[\alpha;\beta]]$

Proof is the same as proof of (9), with "ownb" replaced by "partb."

15) $part[x;y] \wedge x \subset z \Rightarrow part[z;y]$

$(\exists\alpha\epsilon x)[partb[\alpha;y]] \wedge (\forall\alpha\epsilon x)[\alpha\epsilon z] \Rightarrow (\exists\alpha\epsilon z[partb[\alpha;y]]$

Proof is the same as proof of (11) with "ownb" replaced by "partb."

16) $part[x;y] \wedge partg[z;x] \Rightarrow part[z;y]$

$$(\exists\alpha\in x)[\mathrm{partb}[\alpha;y]] \wedge (\forall\beta\in x)(\exists\alpha\in z)[\mathrm{partb}[\alpha;\beta]] \Rightarrow (\exists\alpha\in z)[\mathrm{partb}[\alpha;y]]$$

$$(\exists\alpha)[\alpha\in x \wedge \mathrm{partb}[\alpha;y]] \wedge (\forall\beta)[\beta\in x \Rightarrow (\exists\alpha)[\alpha\in z \wedge \mathrm{partb}[\alpha;\beta]]] \Rightarrow (\exists\alpha)[\alpha\in z \wedge \mathrm{partb}[\alpha;y]]$$

1. $(\exists\alpha)[\alpha\in x \wedge \mathrm{partb}[\alpha;y]] \wedge (\forall\beta)[\beta\in x \Rightarrow (\exists\alpha)[\alpha\in z \wedge \mathrm{partb}[\alpha;\beta]]]$
2. $\gamma\in x \wedge \mathrm{partb}[\gamma;y]$ — ES1
3. $\gamma\in x \Rightarrow (\exists\alpha)[\alpha\in z \wedge \mathrm{partb}[\alpha;\gamma]]$ — US1
4. $(\exists\alpha)[\alpha\in z \wedge \mathrm{partb}[\alpha;\gamma]]$ — 2,3
5. $\mu\in z \wedge \mathrm{partb}[\mu;\gamma]$ — ES5
6. $\mathcal{T}(\mathrm{partp})$ — Axiom
7. $\mathrm{partb}[\mu;\gamma] \wedge \mathrm{partb}[\gamma;y] \Rightarrow \mathrm{partb}[\mu;y]$ — US6
8. $\mu\in z \wedge \mathrm{partb}[\mu;y]$ — 5,2,7
9. $(\exists\alpha)[\alpha\in z \wedge \mathrm{partb}[\alpha;y]]$ — EG8

1.⇒9. qed.

17) $\mathrm{partg}[x;y] \wedge z\in y \Rightarrow \mathrm{part}[x;z]$

$$(\forall\beta\in y)(\exists\alpha\in x)[\mathrm{partb}[\alpha;\beta]] \wedge z\in y \Rightarrow (\exists\alpha\in x)[\mathrm{partb}[\alpha;z]]$$

Proof is the same as proof of (12) with "ownb" replaced by "partb."

<u>Lemma 1</u>: $(\forall\alpha)(\forall\beta)(\forall x)[\mathrm{single}[x] \wedge \alpha\in x \wedge \beta\in x \Rightarrow \alpha=\beta]$

1. $\mathrm{single}[x] \wedge a\in x \wedge b\in x$
2. $(\exists\alpha)[\alpha\in x \wedge (\forall\beta)[\beta\in x \Rightarrow \beta=\alpha]]$ — 1,def.of <u>single</u>
3. $\gamma\in x \wedge (\forall\beta)[\beta\in x \Rightarrow \beta=\gamma]$ — ES2
4. $a\in x \Rightarrow a=\gamma$ — US3
5. $a=\gamma$ — 1,4
6. $b\in x \Rightarrow b=\gamma$ — US3
7. $b=\gamma$ — 1,6
8. $a=b$ — I2-5,7
9. 1.⇒8.

$(\forall\alpha)(\forall\beta)(\forall x)[\mathrm{single}[x] \wedge \alpha\in x \wedge \beta\in x \Rightarrow \alpha=\beta]$ qed. — UG9

18) $\mathrm{right}[x;y] \Rightarrow \sim\mathrm{right}[y;x]$

$$(\exists\alpha\in x)(\exists\beta\in y)[\mathrm{rightb}[\alpha;\beta]] \wedge \mathrm{single}[x] \wedge \mathrm{single}[y]$$
$$\Rightarrow \sim[(\ \alpha\ y)(\ \beta\ x)[\mathrm{rightb}[\alpha;\beta]] \wedge \mathrm{single}[y] \wedge \mathrm{single}[x]]$$

$$(\exists\alpha)[\alpha\in x \wedge (\exists\beta)[\beta\in y \wedge \mathrm{rightb}[\alpha;\beta]]] \wedge \mathrm{single}[x] \wedge \mathrm{single}[y]$$
$$\Rightarrow \sim[(\exists\alpha)[\alpha\in y \wedge (\exists\beta)[\beta\in x \wedge \mathrm{rightb}[\alpha;\beta]]] \wedge \mathrm{single}[y] \wedge \mathrm{single}[x]]$$

1. $(\exists\alpha)[\alpha\in x \wedge (\exists\beta)[\beta\in y \wedge \text{rightb}[\alpha;\beta]]] \wedge \text{single}[x] \wedge \text{single}[y]$
2. $\gamma\in x \wedge (\exists\beta)[\beta\in y \wedge \text{rightb}[\gamma;\beta]]$ ES1
3. $\mu\in y \wedge \text{rightb}[\gamma;\mu]$ ES2
4. $(\exists\alpha)[\alpha\in y \wedge (\exists\beta)[\beta\in x \wedge \text{rightb}[\alpha;\beta]]]$
5. $\omega\in y \wedge (\exists\beta)[\beta\in x \wedge \text{rightb}[\omega;\beta]]$ ES4
6. $\lambda\in x \wedge \text{rightb}[\omega;\lambda]$ ES5
7. $\text{single}[x] \wedge \gamma\in x \wedge \lambda\in x \Rightarrow \gamma=\lambda$ US-Lem.1
8. $\gamma=\lambda$ 1,2,6,7
9. $\text{single}[y] \wedge \mu\in y \wedge \omega\in y \Rightarrow \mu=\omega$ US-Lem.1
10. $\mu=\omega$ 1,3,5,9
11. $\text{rightb}[\lambda;\omega]$ 3,8,10,I2
12. $\mathcal{A}(\text{rightb})$ Axiom
13. $\text{rightb}[\lambda;\omega] \Rightarrow \sim\text{rightb}[\omega;\lambda]$ US12
14. $\sim\text{rightb}[\omega;\lambda]$ 11,13
15. $\text{rightb}[\omega;\lambda]$ 6
16. $\sim 4.$
17. $\sim[4. \wedge \text{single}[y] \wedge \text{single}[x]]$ 16

$1.\Rightarrow 17.$ qed.

19) $\mathcal{T}(\text{right})$

$(\exists\alpha\in x)(\exists\beta\in y)[\text{rightb}[\alpha;\beta]] \wedge (\exists\alpha\in y)(\exists\beta\in z)[\text{rightb}[\alpha;\beta]] \wedge \text{single}[x]$
$\wedge \text{single}[y] \wedge \text{single}[z]$
$\Rightarrow (\exists\alpha\in x)(\exists\beta\in z)[\text{rightb}[\alpha;\beta]] \wedge \text{single}[x] \wedge \text{single}[z]$

$(\exists\alpha)[\alpha\in x \wedge (\exists\beta)[\beta\in y \wedge \text{rightb}[\alpha;\beta]]] \wedge (\exists\alpha)[\alpha\in y \wedge (\exists\beta)[\beta\in z \wedge \text{rightb}[\alpha;\beta]]]$
$\wedge \text{single}[x] \wedge \text{single}[y] \wedge \text{single}[z]$
$\Rightarrow (\exists\alpha)[\alpha\in x \wedge (\exists\beta)[\beta\in z \wedge \text{rightb}[\alpha;\beta]]] \wedge \text{single}[x] \wedge \text{single}[z]$

1. $(\exists\alpha)[\alpha\in x \wedge (\exists\beta)[\beta\in y \wedge \text{rightb}[\alpha;\beta]]] \wedge (\exists\alpha)[\alpha\in y \wedge (\exists\beta)[\beta\in z \wedge \text{rightb}[\alpha;\beta]]]$
$\wedge \text{single}[y]$
2. $\gamma\in x \wedge (\exists\beta)[\beta\in y \wedge \text{rightb}[\gamma;\beta]$ ES1
3. $\mu\in y \wedge \text{rightb}[\gamma;\mu]$ ES2
4. $\omega\in y \wedge (\exists\beta)[\beta\in z \wedge \text{rightb}[\omega;\beta]]$ ES1
5. $\lambda\in y \wedge \text{rightb}[\omega;\beta]$ ES4
6. $\text{single}[y] \wedge \mu\in y \wedge \omega\in y \Rightarrow \mu=\omega$ US-Lem.1
7. $\mu=\omega$ 1,3,4,6
8. $\text{rightb}[\gamma;\omega]$ 3,7,I2
9. $\mathcal{T}(\text{rightb})$ Axiom
10. $\text{rightb}[\gamma;\omega] \wedge \text{rightb}[\omega;\lambda] \Rightarrow \text{rightb}[\gamma;\lambda]$ US9
11. $\lambda\in z \wedge \text{rightb}[\gamma;\lambda]$ 8,5,10
12. $(\exists\beta)[\beta\in z \wedge \text{rightb}[\gamma;\beta]]$ EG11
13. $\gamma\in x \wedge 12.$ 2,12
14. $(\exists\alpha)[\alpha\in x \wedge (\exists\beta)[\beta\in z \wedge \text{rightb}[\alpha;\beta]]]$ EG13
15. $1.\Rightarrow 14.$

$1. \wedge \text{single}[x] \wedge \text{single}[z] \Rightarrow 14. \wedge \text{single}[x] \wedge \text{single}[z]$ qed. 15

20) $jright[x;y] \Rightarrow right[x;y]$

$$(\exists\alpha\epsilon x)(\exists\beta\epsilon y)[jrightb[\alpha;\beta]] \wedge single[x] \wedge single[y] \Rightarrow (\exists\alpha\epsilon x)(\exists\beta\epsilon y)[rightb[\alpha;\beta]] \wedge single[x] \wedge single[y]$$

$$(\exists\alpha)[\alpha\epsilon x \wedge (\exists\beta)[\beta\epsilon y \wedge jrightb[\alpha;\beta]]] \wedge single[x] \wedge single[y] \Rightarrow (\exists\alpha)[\alpha\epsilon x \wedge (\exists\beta)[\beta\epsilon y \wedge rightb[\alpha;\beta]]] \wedge single[x] \wedge single[y]$$

1.	$(\exists\alpha)[\alpha\epsilon x \wedge (\exists\beta)[\beta\epsilon y \wedge jrightb[\alpha;\beta]]]$	
2.	$\gamma\epsilon x \wedge (\exists\beta)[\beta\epsilon y \wedge jrightb[\gamma;\beta]]$	ES1
3.	$\mu\epsilon y \wedge jrightb[\gamma;\mu]$	ES8
4.	$(\forall x)(\forall y)[jrightb[x;y] \Rightarrow rightb[x;y]]$	Axiom
5.	$jrightb[\gamma;\mu] \Rightarrow rightb[\gamma;\mu]$	US4
6.	$\mu\epsilon y \wedge rightb[\gamma;\mu]$	3,5
7.	$(\exists\beta)[\beta\epsilon y \wedge rightb[\gamma;\beta]]$	EG6
8.	$\gamma\epsilon x \wedge 7.$	2,7
9.	$(\exists\alpha)[\alpha\epsilon x \wedge (\exists\beta)[\beta\epsilon y \wedge rightb[\alpha;\beta]]]$	EG8
10.	$1. \Rightarrow 9.$	
	$1. \wedge single[x] \wedge single[y] \Rightarrow 9. \wedge single[x] \wedge single[y]$ qed.	10

21) $jright[x;y] \wedge z \neq y \Rightarrow \sim jright[x;z]$

<u>Note</u>: The SIR programs assumed that "$z \neq y$" was equivalent to the assertion, "the $\underline{z}$ is not the $\underline{y}$." This latter, preferred interpretation can be expressed directly in the SIR1 formalism by

$$single[z] \wedge single[y] \wedge (\forall\alpha\epsilon z)[\alpha\notin y].$$

Therefore the appropriate SIR1 statement corresponding to (21) is:

$$(\exists\alpha\epsilon x)(\exists\beta\epsilon y)[jrightb[\alpha;\beta]] \wedge single[x] \wedge single[y] \wedge single[z] \wedge (\forall\alpha\epsilon z)[\alpha\notin y] \Rightarrow \sim[(\exists\alpha\epsilon x)(\exists\beta\epsilon z)[jrightb[\alpha;\beta]] \wedge single[x] \wedge single[z]]$$

$$(\exists\alpha)[\alpha\epsilon x \wedge (\exists\beta)[\beta\epsilon y \wedge jrightb[\alpha;\beta]]] \wedge single[x] \wedge single[y] \wedge single[z] \wedge (\forall\alpha)[\alpha\epsilon z \Rightarrow \alpha\notin y] \Rightarrow \sim[(\exists\alpha)[\alpha\epsilon x \wedge (\exists\beta)[\beta\epsilon z \wedge jrightb[\alpha;\beta]]] \wedge single[x] \wedge single[z]]$$

Proof is in the proof of (22) below.

22) $jright[x;y] \wedge z \neq x \Rightarrow \sim jright[z;y]$

As discussed in the above note, the appropriate SIR1 statement is:

$$(\exists\alpha\epsilon x)(\exists\beta\epsilon y)[jrightb[\alpha;\beta]] \wedge single[x] \wedge single[y] \wedge single[z] \wedge (\forall\alpha\epsilon z)[\alpha\notin x] \Rightarrow \sim[(\exists\alpha\epsilon z)(\exists\beta\epsilon y)[jrightb[\alpha;\beta]] \wedge single[z] \wedge single[y]]$$

(∃α)[α∈x ∧ (∃β)[β∈y ∧ jrightb[α;β]]] ∧ single[x] ∧ single[y] ∧ single[z]
∧ (∀α)[α∈z ⇒ α∉x]
⇒ ~[(∃α)[α∈z ∧ (∃β)[β∈y ∧ jrightb[α;β]]] ∧ single[z] ∧ single[y]]

1.	(∃α)[α∈x ∧ (∃β)[β∈y ∧ jrightb[α;β]]] ∧ single[x] ∧ single[y]	
2.	λ∈x ∧ (∃β)[β∈y ∧ jrightb[λ;β]]	ES1
3.	ω∈y ∧ jrightb[λ;ω]	ES2
4.	𝒰(jrightb)	Axiom
5.	jrightb[λ;ω] ⇒ (∀α)[[α≠ω ⇒ ~jrightb[λ;α]] ∧[α≠λ ⇒ ~jrightb[α;ω]]]	US4
6.	(∀α)[[α≠ω ⇒ ~jrightb[λ;α]] ∧ [α≠λ ⇒ ~jrightb[α;ω]]]	3,5
7.	single[z] ∧ single[y] ∧ (∀α)[α∈z ⇒ α∉y]	
8.	(∃α)[α∈x ∧ (∃β)[β∈z ∧ jrightb[α;β]]]	
9.	γ∈x ∧ (∃β)[β∈z ∧ jrightb[γ;β]]	ES8
10.	μ∈z ∧ jrightb[γ;μ]	ES9
11.	single[x] ∧ λ∈x ∧ γ∈x ⇒ γ=λ	US-Lem. 1
12.	γ=λ	1,2,9,11
13.	μ∈z ⇒ μ∉y	US7
14.	μ=ω	
15.	μ∉y	10,13
16.	μ∈y	3,14,I2
17.	μ≠ω	
18.	μ≠ω ⇒ ~jrightb[λ;μ]	US6
19.	~jrightb[λ;μ]	17,18
20.	jrightb[λ;μ]	10,12,I2
21.	~8.	
22.	~[8. ∧ single[x] ∧ single[z]]	21,1,7
23.	7. ⇒ 22.	
24.	single[z] ∧ single[x] ∧ (∀α)[α∈z ⇒ α∉x]	
25.	(∃α)[α∈z ∧ (∃β)[β∈y ∧ jrightb[α;β]]]	
26.	a∈z ∧ (∃β)[β∈y ∧ jrightb[a;β]]	ES25
27.	b∈y ∧ jrightb[a;b]	ES26
28.	single[y] ∧ b∈y ∧ ω∈y ⇒ b=ω	US-Lem. 1
29.	b=ω	1,27,3,28
30.	a∈z ⇒ a∉x	US24
31.	a=λ	
32.	a∉x	26,30
33.	a∈x	2,31,I2
34.	a≠λ	
35.	a≠λ ⇒ ~jrightb[a;ω]	US6
36.	~jrightb[a;ω]	34,35
37.	jrightb[a;ω]	26,29,I2
38.	~25.	
39.	~[25. ∧ single[z] ∧ single[y]]	38,24,1
40.	24. ⇒ 39.	
41.	[7 ⇒ 22.] ∧ [24. ⇒ 39.]]	23,40
42.	1. ⇒ [[7. ⇒ 22.] ∧ [24. ⇒ 39.]]	
43.	[1. ⇒ [7. ⇒ 22.]] ∧ [1 ⇒ [24. ⇒ 39.]]	42
	1. ∧ 7. ⇒ 22. qed(21).	43
	1. ∧ 24. ⇒ 39. qed(23).	43

23) $\mathrm{right}[x;y] \wedge \mathrm{right}[y;z] \Rightarrow \sim \mathrm{jright}[x;z]$

$$(\exists\alpha\in x)(\exists\beta\in y)[\mathrm{rightb}[\alpha;\beta]] \wedge (\exists\alpha\in y)(\exists\beta\in z)[\mathrm{rightb}[\alpha;\beta]] \wedge \mathrm{single}[x] \wedge \mathrm{single}[y] \wedge \mathrm{single}[z] \Rightarrow \sim[(\exists\alpha\in x)(\exists\beta\in z)[\mathrm{jrightb}[x;z]] \wedge \mathrm{single}[x] \wedge \mathrm{single}[z]]$$

$$(\exists\alpha)[\alpha\in x \wedge (\exists\beta)[\beta\in y \wedge \mathrm{rightb}[\alpha;\beta]]] \wedge (\exists\alpha)[\alpha\in y \wedge (\exists\beta)[\beta\in z \wedge \mathrm{rightb}[\alpha;\beta]]] \wedge \mathrm{single}[x] \wedge \mathrm{single}[y] \wedge \mathrm{single}[z] \Rightarrow \sim[(\exists\alpha)[\alpha\in x \wedge (\exists\beta)[\beta\in z \wedge \mathrm{jrightb}[\alpha;\beta]]] \wedge \mathrm{single}[x] \wedge \mathrm{single}[z]]$$

1. $(\exists\alpha)[\alpha\in x \wedge (\exists\beta)[\beta\in y \wedge \mathrm{rightb}[\alpha;\beta]]] \wedge (\exists\alpha)[\alpha\in y \wedge (\exists\beta)[\beta\in z \wedge \mathrm{rightb}[\alpha;\beta]]] \wedge \mathrm{single}[x] \wedge \mathrm{single}[y] \wedge \mathrm{single}[z]$
2. $\mu\in x \wedge (\exists\beta)[\beta\in y \wedge \mathrm{rightb}[\mu;\beta]]$ — ES1
3. $\omega\in y \wedge \mathrm{rightb}[\mu;\omega]$ — ES2
4. $\gamma\in y \wedge (\exists\beta)[\beta\in z \wedge \mathrm{rightb}[\gamma;\beta]]$ — ES1
5. $\lambda\in z \wedge \mathrm{rightb}[\gamma;\lambda]]$ — ES4
6. $\mathrm{single}[y] \wedge \omega\in y \wedge \gamma\in y \Rightarrow \omega=\gamma$ — US+Lem.1
7. $\omega=\gamma$ — 1,3,4,6
8. $\mathrm{rightb}[\omega;\lambda]$ — 5,7,I2
9. $(\forall x)(\forall y)(\forall z)[\mathrm{rightb}[x;y] \wedge \mathrm{rightb}[y;z] \Rightarrow \sim\mathrm{jrightb}[x;z]]$ — Axiom
10. $\mathrm{rightb}[\mu;\omega] \wedge \mathrm{rightb}[\omega;\lambda] \Rightarrow \sim\mathrm{jrightb}[\mu;\lambda]$ — US9
11. $(\exists\alpha)[\alpha\in x \wedge (\exists\beta)[\beta\in z \wedge \mathrm{jrightb}[\alpha;\beta]]]$
12. $a\in x \wedge (\exists\beta)[\beta\in z \wedge \mathrm{jrightb}[a;\beta]]$ — ES11
13. $b\in z \wedge \mathrm{jrightb}[a;b]$ — ES12
14. $\mathrm{single}[x] \wedge \mu\in x \wedge a\in x \Rightarrow \mu=a$ — US+Lem.1
15. $\mu=a$ — 1,2,12,14
16. $\mathrm{single}[z] \wedge \lambda\in z \wedge b\in z \Rightarrow \lambda=b$ — US+Lem.1
17. $\lambda=b$ — 1,5,13,16
18. $\mathrm{jrightb}[\mu;\lambda]$ — 13,15,17,I2
19. $\sim\mathrm{jrightb}[\mu;\lambda]$ — 3,8,10
20. $\sim$11.
21. $\sim[$11. $\wedge\ \mathrm{single}[x] \wedge \mathrm{single}[z]]$ — 20

1.$\Rightarrow$21. qed.

a. SET-INCLUSION

```
(THE NEXT SENTENCE IS . .)
(EVERY KEYPUNCH-OPERATOR IS A GIRL)

(THE FUNCTION USED IS . .)
SETR-SELECT
((GENERIC . KEYPUNCH-OPERATOR) (GENERIC . GIRL))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETR
(KEYPUNCH-OPERATOR GIRL)
(ITS REPLY . .)
(I UNDERSTAND THE SUPERSET RELATION BETWEEN GIRL AND KEYPUNCH-OPERATOR)
(I UNDERSTAND THE SUBSET RELATION BETWEEN KEYPUNCH-OPERATOR AND GIRL)


(THE NEXT SENTENCE IS . .)
(ANY GIRL IS AN EXAMPLE OF A PERSON)

(THE FUNCTION USED IS . .)
SETR-SELECT
((GENERIC . GIRL) (GENERIC . PERSON))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETR
(GIRL PERSON)
(ITS REPLY . .)
(I UNDERSTAND THE SUPERSET RELATION BETWEEN PERSON AND GIRL)
(I UNDERSTAND THE SUBSET RELATION BETWEEN GIRL AND PERSON)


(THE NEXT SENTENCE IS . .)
(IS A KEYPUNCH-OPERATOR A PERSON Q)

(THE FUNCTION USED IS . .)
SETRQ-SELECT
((GENERIC . KEYPUNCH-OPERATOR) (GENERIC . PERSON))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETRQ
(KEYPUNCH-OPERATOR PERSON)
(ITS REPLY . .)
YES


(THE NEXT SENTENCE IS . .)
(IS A PERSON A PERSON Q)

(THE FUNCTION USED IS . .)
SETRQ-SELECT
((GENERIC . PERSON) (GENERIC . PERSON))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETRQ
(PERSON PERSON)
(ITS REPLY . .)
YES


(THE NEXT SENTENCE IS . .)
(IS A PERSON A GIRL Q)

(THE FUNCTION USED IS . .)
SETRQ-SELECT
((GENERIC . PERSON) (GENERIC . GIRL))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETRQ
(PERSON GIRL)
(ITS REPLY . .)
SOMETIMES


(THE NEXT SENTENCE IS . .)
(IS A MONKEY A KEYPUNCH-OPERATOR Q)

(THE FUNCTION USED IS . .)
SETRQ-SELECT
((GENERIC . MONKEY) (GENERIC . KEYPUNCH-OPERATOR))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETRQ
(MONKEY KEYPUNCH-OPERATOR)
(ITS REPLY . .)
(INSUFFICIENT INFORMATION)
```

APPENDIX III: FULL-RESPONSE OUTPUT FOR FIGURE 5

b. SET-MEMBERSHIP

```
(THE NEXT SENTENCE IS . .)
(MAX IS AN IBM-7094)

(THE FUNCTION USED IS . .)
SETR-SELECT
((UNIQUE . MAX) (GENERIC . IBM-7094))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETRS
(MAX IBM-7094)
(ITS REPLY . .)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN MAX AND IBM-7094)
(I UNDERSTAND THE MEMBER RELATION BETWEEN IBM-7094 AND MAX)


(THE NEXT SENTENCE IS . .)
(AN IBM-7094 IS A COMPUTER)

(THE FUNCTION USED IS . .)
SETR-SELECT
((GENERIC . IBM-7094) (GENERIC . COMPUTER))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETR
(IBM-7094 COMPUTER)
(ITS REPLY . .)
(I UNDERSTAND THE SUPERSET RELATION BETWEEN COMPUTER AND IBM-7094)
(I UNDERSTAND THE SUBSET RELATION BETWEEN IBM-7094 AND COMPUTER)


(THE NEXT SENTENCE IS . .)
(IS MAX A COMPUTER Q)

(THE FUNCTION USED IS . .)
SETRQ-SELECT
((UNIQUE . MAX) (GENERIC . COMPUTER))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETRSQ


(MAX COMPUTER)
(ITS REPLY . .)
YES


(THE NEXT SENTENCE IS . .)
(THE BOY IS AN MIT-STUDENT)

(THE FUNCTION USED IS . .)
SETR-SELECT
((SPECIFIC . BOY) (GENERIC . MIT-STUDENT))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETRS1
(BOY MIT-STUDENT)
(ITS REPLY . .)
(G02840 IS A BOY)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN G02840 AND BOY)
(I UNDERSTAND THE MEMBER RELATION BETWEEN BOY AND G02840)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN G02840 AND MIT-STUDENT)
(I UNDERSTAND THE MEMBER RELATION BETWEEN MIT-STUDENT AND G02840)
```

```
(THE NEXT SENTENCE IS . .)
(EVERY MIT-STUDENT IS A BRIGHT-PERSON)

(THE FUNCTION USED IS . .)
SETR-SELECT
((GENERIC . MIT-STUDENT) (GENERIC . BRIGHT-PERSON))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETR
(MIT-STUDENT BRIGHT-PERSON)
(ITS REPLY . .)
(I UNDERSTAND THE SUPERSET RELATION BETWEEN BRIGHT-PERSON AND MIT-STUDENT)
(I UNDERSTAND THE SUBSET RELATION BETWEEN MIT-STUDENT AND BRIGHT-PERSON)


(THE NEXT SENTENCE IS . .)
(IS THE BOY A BRIGHT-PERSON Q)

(THE FUNCTION USED IS . .)
SETRQ-SELECT
((SPECIFIC . BOY) (GENERIC . BRIGHT-PERSON))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETRS1Q
(BOY BRIGHT-PERSON)
(ITS REPLY . .)
YES


(THE NEXT SENTENCE IS . .)
(JOHN IS A BOY)

(THE FUNCTION USED IS . .)
SETR-SELECT
((UNIQUE . JOHN) (GENERIC . BOY))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETRS


(JOHN BOY)
(ITS REPLY . .)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN JOHN AND BOY)
(I UNDERSTAND THE MEMBER RELATION BETWEEN BOY AND JOHN)


(THE NEXT SENTENCE IS . .)
(IS THE BOY A BRIGHT-PERSON Q)

(THE FUNCTION USED IS . .)
SETRQ-SELECT
((SPECIFIC . BOY) (GENERIC . BRIGHT-PERSON))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETRS1Q
(BOY BRIGHT-PERSON)
(ITS REPLY . .)
(WHICH BOY . . (G02840 JOHN))
```

```
(THE NEXT SENTENCE IS . .)
(THE MAN IS A JERK)

(THE FUNCTION USED IS . .)
SETR-SELECT
((SPECIFIC . MAN) (GENERIC . JERK))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETRSI
(MAN JERK)
(ITS REPLY . .)
(G02840 IS A MAN)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN G02840 AND MAN)
(I UNDERSTAND THE MEMBER RELATION BETWEEN MAN AND G02840)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN G02840 AND JERK)
(I UNDERSTAND THE MEMBER RELATION BETWEEN JERK AND G02840)


(THE NEXT SENTENCE IS . .)
(JACK IS A DOPE)

(THE FUNCTION USED IS . .)
SETR-SELECT
((UNIQUE . JACK) (GENERIC . DOPE))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETRS
(JACK DOPE)
(ITS REPLY . .)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN JACK AND DOPE)
(I UNDERSTAND THE MEMBER RELATION BETWEEN DOPE AND JACK)


(THE NEXT SENTENCE IS . .)
(JOHN IS JACK)

(THE FUNCTION USED IS . .)
SETR-SELECT
((UNIQUE . JOHN) (UNIQUE . JACK))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
EQUIV
(JOHN JACK)
(ITS REPLY . .)
(I UNDERSTAND THE EQUIV RELATION BETWEEN JOHN AND JACK)
(I UNDERSTAND THE EQUIV RELATION BETWEEN JACK AND JOHN)


(THE NEXT SENTENCE IS . .)
(IS JOHN A DOPE Q)

(THE FUNCTION USED IS . .)
SETRQ-SELECT
((UNIQUE . JOHN) (GENERIC . DOPE))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETRSQ
(JOHN DOPE)
(ITS REPLY . .)
YES

(THE NEXT SENTENCE IS . .)
(IS THE MAN A DOPE Q)

(THE FUNCTION USED IS . .)
SETRQ-SELECT
((SPECIFIC . MAN) (GENERIC . DOPE))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETRSIQ
(MAN DOPE)
(ITS REPLY . .)
(INSUFFICIENT INFORMATION)


(THE NEXT SENTENCE IS . .)
(JOHN IS THE MAN)

(THE FUNCTION USED IS . .)
SETR-SELECT
((UNIQUE . JOHN) (SPECIFIC . MAN))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
EQUIV1
(JOHN MAN)
(ITS REPLY . .)
(I UNDERSTAND THE EQUIV RELATION BETWEEN JOHN AND G02840)
(I UNDERSTAND THE EQUIV RELATION BETWEEN G02840 AND JOHN)


(THE NEXT SENTENCE IS . .)
(IS THE MAN A DOPE Q)

(THE FUNCTION USED IS . .)
SETRQ-SELECT
((SPECIFIC . MAN) (GENERIC . DOPE))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETRSIQ
(MAN DOPE)
(ITS REPLY . .)
YES


(THE NEXT SENTENCE IS . .)
(JIM IS A MAN)

(THE FUNCTION USED IS . .)
SETR-SELECT
((UNIQUE . JIM) (GENERIC . MAN))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETRS
(JIM MAN)
(ITS REPLY . .)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN JIM AND MAN)
(I UNDERSTAND THE MEMBER RELATION BETWEEN MAN AND JIM)

(THE NEXT SENTENCE IS . .)
(IS THE MAN A DOPE Q)

(THE FUNCTION USED IS . .)
SETRQ-SELECT
((SPECIFIC . MAN) (GENERIC . DOPE))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETRSIQ
(MAN DOPE)
(ITS REPLY . .)
(WHICH MAN . . (G02840 JIM))
```

```
(THE NEXT SENTENCE IS . .)
(EVERY FIREMAN OWNS A PAIR-OF-RED-SUSPENDERS)

(THE FUNCTION USED IS . .)
OWN-SELECT
((GENERIC . PAIR-OF-RED-SUSPENDERS) (GENERIC . FIREMAN))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
OWNR
(PAIR-OF-RED-SUSPENDERS FIREMAN)
(ITS REPLY . .)
(I UNDERSTAND THE POSSESS-BY-EACH RELATION BETWEEN PAIR-OF-RED-SUSPENDERS AND FIREMAN)
(I UNDERSTAND THE OWNED-BY-EACH RELATION BETWEEN FIREMAN AND PAIR-OF-RED-SUSPENDERS)


(THE NEXT SENTENCE IS . .)
(DOES A PAIR-OF-RED-SUSPENDERS OWN A PAIR-OF-RED-SUSPENDERS Q)

(THE FUNCTION USED IS . .)
OWNQ-SELECT
((GENERIC . PAIR-OF-RED-SUSPENDERS) (GENERIC . PAIR-OF-RED-SUSPENDERS))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
OWNRQ
(PAIR-OF-RED-SUSPENDERS PAIR-OF-RED-SUSPENDERS)
(ITS REPLY . .)
(NO ** THEY ARE THE SAME)


(THE NEXT SENTENCE IS . .)
(DOES A DOCTOR OWN A PAIR-OF-RED-SUSPENDERS Q)

(THE FUNCTION USED IS . .)
OWNQ-SELECT
((GENERIC . PAIR-OF-RED-SUSPENDERS) (GENERIC . DOCTOR))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
OWNRQ
(PAIR-OF-RED-SUSPENDERS DOCTOR)
(ITS REPLY . .)
(INSUFFICIENT INFORMATION)


(THE NEXT SENTENCE IS . .)
(A FIRECHIEF IS A FIREMAN)

(THE FUNCTION USED IS . .)
SETR-SELECT
((GENERIC . FIRECHIEF) (GENERIC . FIREMAN))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETR
(FIRECHIEF FIREMAN)
(ITS REPLY . .)
(I UNDERSTAND THE SUPERSET RELATION BETWEEN FIREMAN AND FIRECHIEF)
(I UNDERSTAND THE SUBSET RELATION BETWEEN FIRECHIEF AND FIREMAN)


(THE NEXT SENTENCE IS . .)
(DOES A FIRECHIEF OWN A PAIR-OF-RED-SUSPENDERS Q)

(THE FUNCTION USED IS . .)
OWNQ-SELECT
((GENERIC . PAIR-OF-RED-SUSPENDERS) (GENERIC . FIRECHIEF))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
OWNRQ
(PAIR-OF-RED-SUSPENDERS FIRECHIEF)
(ITS REPLY . .)
YES
```

d. OWNERSHIP, GENERAL

e. OWNERSHIP, SPECIFIC

```
(THE NEXT SENTENCE IS . .)
(ALFRED OWNS A LOG-LOG-DECITRIG)

(THE FUNCTION USED IS . .)
OWN-SELECT
((GENERIC . LOG-LOG-DECITRIG) (UNIQUE . ALFRED))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
OWNRGU
(LOG-LOG-DECITRIG ALFRED)
(ITS REPLY . .)
(I UNDERSTAND THE POSSESS RELATION BETWEEN LOG-LOG-DECITRIG AND ALFRED)
(I UNDERSTAND THE OWNED RELATION BETWEEN ALFRED AND LOG-LOG-DECITRIG)

(THE NEXT SENTENCE IS . .)
(A LOG-LOG-DECITRIG IS A SLIDE-RULE)

(THE FUNCTION USED IS . .)
SETR-SELECT
((GENERIC . LOG-LOG-DECITRIG) (GENERIC . SLIDE-RULE))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETR
(LOG-LOG-DECITRIG SLIDE-RULE)
(ITS REPLY . .)
(I UNDERSTAND THE SUPERSET RELATION BETWEEN SLIDE-RULE AND LOG-LOG-DECITRIG)
(I UNDERSTAND THE SUBSET RELATION BETWEEN LOG-LOG-DECITRIG AND SLIDE-RULE)

(THE NEXT SENTENCE IS . .)
(DOES ALFRED OWN A SLIDE-RULE Q)

(THE FUNCTION USED IS . .)
OWNQ-SELECT
((GENERIC . SLIDE-RULE) (UNIQUE . ALFRED))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
OWNRGUQ
(SLIDE-RULE ALFRED)
(ITS REPLY . .)
YES

(THE NEXT SENTENCE IS . .)
(EVERY ENGINEERING-STUDENT OWNS A SLIDE-RULE)

(THE FUNCTION USED IS . .)
OWN-SELECT
((GENERIC . SLIDE-RULE) (GENERIC . ENGINEERING-STUDENT))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
OWNR
(SLIDE-RULE ENGINEERING-STUDENT)
(ITS REPLY . .)
(I UNDERSTAND THE POSSESS-BY-EACH RELATION BETWEEN SLIDE-RULE AND ENGINEERING-STUDENT)
(I UNDERSTAND THE OWNED-BY-EACH RELATION BETWEEN ENGINEERING-STUDENT AND SLIDE-RULE)

(THE NEXT SENTENCE IS . .)
(VERNON IS A TECH-MAN)

(THE FUNCTION USED IS . .)
SETR-SELECT
((UNIQUE . VERNON) (GENERIC . TECH-MAN))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETRS
(VERNON TECH-MAN)
(ITS REPLY . .)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN VERNON AND TECH-MAN)
(I UNDERSTAND THE MEMBER RELATION BETWEEN TECH-MAN AND VERNON)

(THE NEXT SENTENCE IS . .)
(A TECH-MAN IS AN ENGINEERING-STUDENT)

(THE FUNCTION USED IS . .)
SETR-SELECT
((GENERIC . TECH-MAN) (GENERIC . ENGINEERING-STUDENT))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETR
(TECH-MAN ENGINEERING-STUDENT)
(ITS REPLY . .)
(I UNDERSTAND THE SUPERSET RELATION BETWEEN ENGINEERING-STUDENT AND TECH-MAN)
(I UNDERSTAND THE SUBSET RELATION BETWEEN TECH-MAN AND ENGINEERING-STUDENT)

(THE NEXT SENTENCE IS . .)
(DOES VERNON OWN A SLIDE-RULE Q)

(THE FUNCTION USED IS . .)
OWNQ-SELECT
((GENERIC . SLIDE-RULE) (UNIQUE . VERNON))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
OWNRGUQ
(SLIDE-RULE VERNON)
(ITS REPLY . .)
YES

(THE NEXT SENTENCE IS . .)
(DOES AN ENGINEERING-STUDENT OWN THE LOG-LOG-DECITRIG Q)

(THE FUNCTION USED IS . .)
OWNQ-SELECT
((SPECIFIC . LOG-LOG-DECITRIG) (GENERIC . ENGINEERING-STUDENT))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
OWNRSGQ
(LOG-LOG-DECITRIG ENGINEERING-STUDENT)
(ITS REPLY . .)
(G02840 IS A LOG-LOG-DECITRIG)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN G02840 AND LOG-LOG-DECITRIG)
(I UNDERSTAND THE MEMBER RELATION BETWEEN LOG-LOG-DECITRIG AND G02840)
(INSUFFICIENT INFORMATION)

(THE NEXT SENTENCE IS . .)
(ALFRED IS A TECH-MAN)

(THE FUNCTION USED IS . .)
SETR-SELECT
((UNIQUE . ALFRED) (GENERIC . TECH-MAN))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETRS
(ALFRED TECH-MAN)
(ITS REPLY . .)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN ALFRED AND TECH-MAN)
(I UNDERSTAND THE MEMBER RELATION BETWEEN TECH-MAN AND ALFRED)

(THE NEXT SENTENCE IS . .)
(DOES AN ENGINEERING-STUDENT OWN THE LOG-LOG-DECITRIG Q)

(THE FUNCTION USED IS . .)
OWNQ-SELECT
((SPECIFIC . LOG-LOG-DECITRIG) (GENERIC . ENGINEERING-STUDENT))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
OWNRSGQ
(LOG-LOG-DECITRIG ENGINEERING-STUDENT)
(ITS REPLY . .)
YES
```

f. PART-WHOLE, GENERAL

```
(THE NEXT SENTENCE IS . .)
(A NOSE IS PART OF A PERSON)

(THE FUNCTION USED IS . .)
PARTR-SELECT
((GENERIC . NOSE) (GENERIC . PERSON))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
PARTR
(NOSE PERSON)
(ITS REPLY . .)
(I UNDERSTAND THE SUBPART-OF-EACH RELATION BETWEEN NOSE AND PERSON)
(I UNDERSTAND THE SUPERPART-OF-EACH RELATION BETWEEN PERSON AND NOSE)


(THE NEXT SENTENCE IS . .)
(A NOSTRIL IS A PART OF A NOSE)

(THE FUNCTION USED IS . .)
PARTR-SELECT
((GENERIC . NOSTRIL) (GENERIC . NOSE))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
PARTR
(NOSTRIL NOSE)
(ITS REPLY . .)
(I UNDERSTAND THE SUBPART-OF-EACH RELATION BETWEEN NOSTRIL AND NOSE)
(I UNDERSTAND THE SUPERPART-OF-EACH RELATION BETWEEN NOSE AND NOSTRIL)


(THE NEXT SENTENCE IS . .)
(A PROFESSOR IS A TEACHER)

(THE FUNCTION USED IS . .)
SETR-SELECT
((GENERIC . PROFESSOR) (GENERIC . TEACHER))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETR

(PROFESSOR TEACHER)
(ITS REPLY . .)
(I UNDERSTAND THE SUPERSET RELATION BETWEEN TEACHER AND PROFESSOR)
(I UNDERSTAND THE SUBSET RELATION BETWEEN PROFESSOR AND TEACHER)


(THE NEXT SENTENCE IS . .)
(A TEACHER IS A PERSON)

(THE FUNCTION USED IS . .)
SETR-SELECT
((GENERIC . TEACHER) (GENERIC . PERSON))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETR
(TEACHER PERSON)
(ITS REPLY . .)
(I UNDERSTAND THE SUPERSET RELATION BETWEEN PERSON AND TEACHER)
(I UNDERSTAND THE SUBSET RELATION BETWEEN TEACHER AND PERSON)
```

```
(THE NEXT SENTENCE IS . .)
(IS A NOSTRIL PART OF A PROFESSOR Q)

(THE FUNCTION USED IS . .)
PARTRQ-SELECT
((GENERIC . NOSTRIL) (GENERIC . PROFESSOR))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
PARTRQ
(NOSTRIL PROFESSOR)
(ITS REPLY . .)
YES


(THE NEXT SENTENCE IS . .)
(IS A NOSE PART OF A NOSE Q)

(THE FUNCTION USED IS . .)
PARTRQ-SELECT
((GENERIC . NOSE) (GENERIC . NOSE))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
PARTRQ
(NOSE NOSE)
(ITS REPLY . .)
(NO , PART MEANS PROPER SUBPART)


(THE NEXT SENTENCE IS . .)
(A PERSON IS A LIVING-CREATURE)

(THE FUNCTION USED IS . .)
SETR-SELECT
((GENERIC . PERSON) (GENERIC . LIVING-CREATURE))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETR
(PERSON LIVING-CREATURE)
(ITS REPLY . .)
(I UNDERSTAND THE SUPERSET RELATION BETWEEN LIVING-CREATURE AND PERSON)
(I UNDERSTAND THE SUBSET RELATION BETWEEN PERSON AND LIVING-CREATURE)


(THE NEXT SENTENCE IS . .)
(IS A NOSTRIL PART OF A LIVING-CREATURE Q)

(THE FUNCTION USED IS . .)
PARTRQ-SELECT
((GENERIC . NOSTRIL) (GENERIC . LIVING-CREATURE))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
PARTRQ
(NOSTRIL LIVING-CREATURE)
(ITS REPLY . .)
SOMETIMES


(THE NEXT SENTENCE IS . .)
(IS A LIVING-CREATURE PART OF A NOSE Q)

(THE FUNCTION USED IS . .)
PARTRQ-SELECT
((GENERIC . LIVING-CREATURE) (GENERIC . NOSE))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
PARTRQ
(LIVING-CREATURE NOSE)
(ITS REPLY . .)
(NO , NOSE IS SOMETIMES PART OF LIVING-CREATURE)
```

```
(THE NEXT SENTENCE IS . .)
(A VAN-DYKE IS PART OF FERREN)

(THE FUNCTION USED IS . .)
PARTR-SELECT
((GENERIC . VAN-DYKE) (UNIQUE . FERREN))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
PARTRGU
(VAN-DYKE FERREN)
(ITS REPLY . .)
(I UNDERSTAND THE SUBPART RELATION BETWEEN VAN-DYKE AND FERREN)
(I UNDERSTAND THE SUPERPART RELATION BETWEEN FERREN AND VAN-DYKE)

(THE NEXT SENTENCE IS . .)
(A VAN-DYKE IS A BEARD)

(THE FUNCTION USED IS . .)
SETR-SELECT
((GENERIC . VAN-DYKE) (GENERIC . BEARD))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETR
(VAN-DYKE BEARD)
(ITS REPLY . .)
(I UNDERSTAND THE SUPERSET RELATION BETWEEN BEARD AND VAN-DYKE)
(I UNDERSTAND THE SUBSET RELATION BETWEEN VAN-DYKE AND BEARD)

(THE NEXT SENTENCE IS . .)
(IS A BEARD PART OF FERREN Q)

(THE FUNCTION USED IS . .)
PARTRQ-SELECT
((GENERIC . BEARD) (UNIQUE . FERREN))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)

PARTRGUQ
(BEARD FERREN)
(ITS REPLY . .)
YES

(THE NEXT SENTENCE IS . .)
(A CRT IS A DISPLAY-DEVICE)

(THE FUNCTION USED IS . .)
SETR-SELECT
((GENERIC . CRT) (GENERIC . DISPLAY-DEVICE))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETR
(CRT DISPLAY-DEVICE)
(ITS REPLY . .)
(I UNDERSTAND THE SUPERSET RELATION BETWEEN DISPLAY-DEVICE AND CRT)
(I UNDERSTAND THE SUBSET RELATION BETWEEN CRT AND DISPLAY-DEVICE)

(THE NEXT SENTENCE IS . .)
(A CRT IS PART OF THE PDP-1)

(THE FUNCTION USED IS . .)
PARTR-SELECT
((GENERIC . CRT) (SPECIFIC . PDP-1))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
PARTRGS
(CRT PDP-1)
(ITS REPLY . .)
(G02840 IS A PDP-1)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN G02840 AND PDP-1)
(I UNDERSTAND THE MEMBER RELATION BETWEEN PDP-1 AND G02840)
(I UNDERSTAND THE SUBPART RELATION BETWEEN CRT AND G02840)
(I UNDERSTAND THE SUPERPART RELATION BETWEEN G02840 AND CRT)

(THE NEXT SENTENCE IS . .)
(SAM IS THE PDP-1)

(THE FUNCTION USED IS . .)
SETR-SELECT
((UNIQUE . SAM) (SPECIFIC . PDP-1))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
EQUIV1
(SAM PDP-1)
(ITS REPLY . .)
(I UNDERSTAND THE EQUIV RELATION BETWEEN SAM AND G02840)
(I UNDERSTAND THE EQUIV RELATION BETWEEN G02840 AND SAM)
```

g. PART-WHOLE, SPECIFIC

```
(THE NEXT SENTENCE IS . .)
(A SCREEN IS PART OF EVERY DISPLAY-DEVICE)

(THE FUNCTION USED IS . .)
PARTR-SELECT
((GENERIC . SCREEN) (GENERIC . DISPLAY-DEVICE))
(THE REPLY . .)



(THE SUB-FUNCTION USED IS . .)
PARTR
(SCREEN DISPLAY-DEVICE)
(ITS REPLY . .)
(I UNDERSTAND THE SUBPART-OF-EACH RELATION BETWEEN SCREEN AND DISPLAY-DEVICE)
(I UNDERSTAND THE SUPERPART-OF-EACH RELATION BETWEEN DISPLAY-DEVICE AND SCREEN)


(THE NEXT SENTENCE IS . .)
(IS A SCREEN PART OF SAM Q)

(THE FUNCTION USED IS . .)
PARTRQ-SELECT
((GENERIC . SCREEN) (UNIQUE . SAM))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
PARTRGUQ
(SCREEN SAM)
(ITS REPLY . .)
YES


(THE NEXT SENTENCE IS . .)
(A BEARD IS PART OF A BEATNIK)

(THE FUNCTION USED IS . .)
PARTR-SELECT
((GENERIC . BEARD) (GENERIC . BEATNIK))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
PARTR
(BEARD BEATNIK)
(ITS REPLY . .)
(I UNDERSTAND THE SUBPART-OF-EACH RELATION BETWEEN BEARD AND BEATNIK)
(I UNDERSTAND THE SUPERPART-OF-EACH RELATION BETWEEN BEATNIK AND BEARD)


(THE NEXT SENTENCE IS . .)
(EVERY COFFEE-HOUSE-CUSTOMER IS A BEATNIK)

(THE FUNCTION USED IS . .)
SETR-SELECT
((GENERIC . COFFEE-HOUSE-CUSTOMER) (GENERIC . BEATNIK))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETR
(COFFEE-HOUSE-CUSTOMER BEATNIK)
(ITS REPLY . .)
(I UNDERSTAND THE SUPERSET RELATION BETWEEN BEATNIK AND COFFEE-HOUSE-CUSTOMER)
(I UNDERSTAND THE SUBSET RELATION BETWEEN COFFEE-HOUSE-CUSTOMER AND BEATNIK)


(THE NEXT SENTENCE IS . .)
(BUZZ IS A COFFEE-HOUSE-CUSTOMER)

(THE FUNCTION USED IS . .)
SETR-SELECT
((UNIQUE . BUZZ) (GENERIC . COFFEE-HOUSE-CUSTOMER))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETRS



(BUZZ COFFEE-HOUSE-CUSTOMER)
(ITS REPLY . .)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN BUZZ AND COFFEE-HOUSE-CUSTOMER)
(I UNDERSTAND THE MEMBER RELATION BETWEEN COFFEE-HOUSE-CUSTOMER AND BUZZ)


(THE NEXT SENTENCE IS . .)
(IS A BEARD PART OF BUZZ Q)

(THE FUNCTION USED IS . .)
PARTRQ-SELECT
((GENERIC . BEARD) (UNIQUE . BUZZ))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
PARTRGUQ
(BEARD BUZZ)
(ITS REPLY . .)
YES
```

g. PART-WHOLE, SPECIFIC (Cont.)

h. NUMBER

```
(THE NEXT SENTENCE IS . .)
(A BOY IS A PERSON)

(THE FUNCTION USED IS . .)
SETR-SELECT
((GENERIC . BOY) (GENERIC . PERSON))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETR
(BOY PERSON)
(ITS REPLY . .)
(I UNDERSTAND THE SUPERSET RELATION BETWEEN PERSON AND BOY)
(I UNDERSTAND THE SUBSET RELATION BETWEEN BOY AND PERSON)


(THE NEXT SENTENCE IS . .)
(JOHN IS A BOY)

(THE FUNCTION USED IS . .)
SETR-SELECT
((UNIQUE . JOHN) (GENERIC . BOY))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETRS
(JOHN BOY)
(ITS REPLY . .)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN JOHN AND BOY)
(I UNDERSTAND THE MEMBER RELATION BETWEEN BOY AND JOHN)


(THE NEXT SENTENCE IS . .)
(A FINGER IS PART OF A HAND)

(THE FUNCTION USED IS . .)
PARTR-SELECT
((GENERIC . FINGER) (GENERIC . HAND))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)


PARTR
(FINGER HAND)
(ITS REPLY . .)
(I UNDERSTAND THE SUBPART-OF-EACH RELATION BETWEEN FINGER AND HAND)
(I UNDERSTAND THE SUPERPART-OF-EACH RELATION BETWEEN HAND AND FINGER)


(THE NEXT SENTENCE IS . .)
(HOW MANY FINGERS DOES JOHN HAVE Q)

(THE FUNCTION USED IS . .)
HAVE-RESOLVE
(FINGER (UNIQUE . JOHN))
(THE REPLY . .)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(I DON*T KNOW WHETHER FINGER IS PART OF JOHN)

(THE NEXT SENTENCE IS . .)
(THERE IS ONE HAND ON EACH ARM)

(THE FUNCTION USED IS . .)
PARTRN-SELECT
((GENERIC . ARM) (1 . HAND))
(THE REPLY . .)
(I UNDERSTAND THE SUPERPART-OF-EACH RELATION BETWEEN ARM AND HAND)
(I REALIZE THE NUMBER RELATION BETWEEN 1 AND (PLIST NAME ARM))
(I UNDERSTAND THE SUBPART-OF-EACH RELATION BETWEEN HAND AND ARM)
(I REALIZE THE NUMBER RELATION BETWEEN 1 AND (PLIST NAME HAND))


(THE NEXT SENTENCE IS . .)
(THERE ARE TWO ARMS ON A PERSON)

(THE FUNCTION USED IS . .)
PARTRN-SELECT
((GENERIC . PERSON) (2 . ARM))
(THE REPLY . .)
(I UNDERSTAND THE SUPERPART-OF-EACH RELATION BETWEEN PERSON AND ARM)
(I REALIZE THE NUMBER RELATION BETWEEN 2 AND (PLIST NAME PERSON))
(I UNDERSTAND THE SUBPART-OF-EACH RELATION BETWEEN ARM AND PERSON)
(I REALIZE THE NUMBER RELATION BETWEEN 2 AND (PLIST NAME ARM))


(THE NEXT SENTENCE IS . .)
(HOW MANY FINGERS DOES JOHN HAVE Q)

(THE FUNCTION USED IS . .)
HAVE-RESOLVE
(FINGER (UNIQUE . JOHN))
(THE REPLY . .)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(I KNOW THE SUPERPART-OF-EACH RELATION BETWEEN HAND AND FINGER)
((HOW MANY FINGER PER HAND Q))


(THE NEXT SENTENCE IS . .)
(A HAND HAS 5 FINGERS)

(THE FUNCTION USED IS . .)
HASN-RESOLVE
((5 . FINGER) (GENERIC . HAND))
(THE REPLY . .)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(I KNOW THE SUPERPART-OF-EACH RELATION BETWEEN HAND AND FINGER)
(I REALIZE THE NUMBER RELATION BETWEEN 5 AND (PLIST NAME HAND))
(I KNOW THE SUBPART-OF-EACH RELATION BETWEEN FINGER AND HAND)
(I REALIZE THE NUMBER RELATION BETWEEN 5 AND (PLIST NAME FINGER))


(THE NEXT SENTENCE IS . .)
(HOW MANY FINGERS DOES JOHN HAVE Q)

(THE FUNCTION USED IS . .)
HAVE-RESOLVE
(FINGER (UNIQUE . JOHN))
(THE REPLY . .)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(I KNOW THE SUPERPART-OF-EACH RELATION BETWEEN HAND AND FINGER)
(I KNOW THE SUPERPART-OF-EACH RELATION BETWEEN ARM AND HAND)
(I KNOW THE SUPERPART-OF-EACH RELATION BETWEEN PERSON AND ARM)
(THE ANSWER IS 10)
```

I. LEFT-TO-RIGHT POSITION

```
(THE NEXT SENTENCE IS . .)
(THE TELEPHONE IS JUST TO THE RIGHT OF THE BOOK)

(THE FUNCTION USED IS . .)
JRIGHT-SELECT
((SPECIFIC . TELEPHONE) (SPECIFIC . BOOK))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
JRIGHT
(TELEPHONE BOOK)
(ITS REPLY . .)
(G02840 IS A TELEPHONE)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN G02840 AND TELEPHONE)
(I UNDERSTAND THE MEMBER RELATION BETWEEN TELEPHONE AND G02840)
(G02841 IS A BOOK)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN G02841 AND BOOK)
(I UNDERSTAND THE MEMBER RELATION BETWEEN BOOK AND G02841)
(I REALIZE THE JRIGHT RELATION BETWEEN TELEPHONE AND BOOK)
(I REALIZE THE JLEFT RELATION BETWEEN BOOK AND TELEPHONE)

(THE NEXT SENTENCE IS . .)
(THE TELEPHONE IS JUST TO THE LEFT OF THE PAD)


(THE FUNCTION USED IS . .)
JRIGHT-SELECT
((SPECIFIC . PAD) (SPECIFIC . TELEPHONE))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
JRIGHT
(PAD TELEPHONE)
(ITS REPLY . .)
(G02842 IS A PAD)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN G02842 AND PAD)
(I UNDERSTAND THE MEMBER RELATION BETWEEN PAD AND G02842)
(I REALIZE THE JRIGHT RELATION BETWEEN PAD AND TELEPHONE)
(I REALIZE THE JLEFT RELATION BETWEEN TELEPHONE AND PAD)

(THE NEXT SENTENCE IS . .)
(IS THE PAD JUST TO THE RIGHT OF THE BOOK Q)

(THE FUNCTION USED IS . .)
JRIGHTQ-SELECT
((SPECIFIC . PAD) (SPECIFIC . BOOK))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
JRIGHTSSQ
(PAD BOOK)
(ITS REPLY . .)
NO

(THE NEXT SENTENCE IS . .)
(IS THE BOOK TO THE LEFT OF THE PAD Q)

(THE FUNCTION USED IS . .)
RIGHTQ-SELECT
((SPECIFIC . PAD) (SPECIFIC . BOOK))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
RIGHTSSQ
(PAD BOOK)
(ITS REPLY . .)
YES
```

```
(THE NEXT SENTENCE IS . .)
(THE PAD IS TO THE RIGHT OF THE TELEPHONE)

(THE FUNCTION USED IS . .)
RIGHT-SELECT
((SPECIFIC . PAD) (SPECIFIC . TELEPHONE))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
RIGHT
(PAD TELEPHONE)
(ITS REPLY . .)
(THE ABOVE STATEMENT IS ALREADY KNOWN)


(THE NEXT SENTENCE IS . .)
(THE PAD IS TO THE LEFT OF THE TELEPHONE)

(THE FUNCTION USED IS . .)

RIGHT-SELECT
((SPECIFIC . TELEPHONE) (SPECIFIC . PAD))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
RIGHT
(TELEPHONE PAD)
(ITS REPLY . .)
(THE ABOVE STATEMENT IS IMPOSSIBLE)


(THE NEXT SENTENCE IS . .)
(THE ASH-TRAY IS TO THE LEFT OF THE BOOK)

(THE FUNCTION USED IS . .)
RIGHT-SELECT
((SPECIFIC . BOOK) (SPECIFIC . ASH-TRAY))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
RIGHT
(BOOK ASH-TRAY)
(ITS REPLY . .)
(G02843 IS A ASH-TRAY)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN G02843 AND ASH-TRAY)
(I UNDERSTAND THE MEMBER RELATION BETWEEN ASH-TRAY AND G02843)
(I UNDERSTAND THE RIGHT RELATION BETWEEN BOOK AND ASH-TRAY)
(I UNDERSTAND THE LEFT RELATION BETWEEN ASH-TRAY AND BOOK)

(THE NEXT SENTENCE IS . .)
(THE PENCIL IS TO THE LEFT OF THE PAD)

(THE FUNCTION USED IS . .)
RIGHT-SELECT
((SPECIFIC . PAD) (SPECIFIC . PENCIL))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
RIGHT
(PAD PENCIL)
(ITS REPLY . .)
(G02844 IS A PENCIL)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN G02844 AND PENCIL)
(I UNDERSTAND THE MEMBER RELATION BETWEEN PENCIL AND G02844)
(I UNDERSTAND THE RIGHT RELATION BETWEEN PAD AND PENCIL)
(I UNDERSTAND THE LEFT RELATION BETWEEN PENCIL AND PAD)
```

i. LEFT-TO-RIGHT POSITION (Cont.)

```
(THE NEXT SENTENCE IS . .)
(THE PAPER IS TO THE RIGHT OF THE TELEPHONE)

(THE FUNCTION USED IS . .)
RIGHT-SELECT
((SPECIFIC . PAPER) (SPECIFIC . TELEPHONE))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
RIGHT
(PAPER TELEPHONE)
(ITS REPLY . .)
(G02845 IS A PAPER)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN G02845 AND PAPER)
(I UNDERSTAND THE MEMBER RELATION BETWEEN PAPER AND G02845)
(I UNDERSTAND THE RIGHT RELATION BETWEEN PAPER AND TELEPHONE)


(I UNDERSTAND THE LEFT RELATION BETWEEN TELEPHONE AND PAPER)


(THE NEXT SENTENCE IS . .)
(WHERE IS THE PAD Q)

(THE FUNCTION USED IS . .)
WHERE-SELECT
((SPECIFIC . PAD))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
WHERES
(PAD)
(ITS REPLY . .)
(JUST TO THE RIGHT OF THE TELEPHONE)
(SOMEWHERE TO THE RIGHT OF THE FOLLOWING . . (PENCIL))


(THE NEXT SENTENCE IS . .)
(WHAT IS THE POSITION OF THE PAD Q)

(THE FUNCTION USED IS . .)
LOC-SELECT
((SPECIFIC . PAD))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
LOCATES
(PAD)
(ITS REPLY . .)
(THE LEFT-TO-RIGHT ORDER IS AS FOLLOWS)
(ASH-TRAY (BOOK TELEPHONE PAD) PAPER)
(TO FURTHER SPECIFY THE POSITIONS YOU MUST INDICATE WHERE THE PENCIL IS WITH RESPECT TO THE ASH-TRAY)

(THE NEXT SENTENCE IS . .)
(THE BOOK IS JUST TO THE RIGHT OF THE ASH-TRAY)

(THE FUNCTION USED IS . .)
JRIGHT-SELECT
((SPECIFIC . BOOK) (SPECIFIC . ASH-TRAY))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
JRIGHT
(BOOK ASH-TRAY)
(ITS REPLY . .)
(I REALIZE THE JRIGHT RELATION BETWEEN BOOK AND ASH-TRAY)
(I REALIZE THE JLEFT RELATION BETWEEN ASH-TRAY AND BOOK)
```

```
(THE NEXT SENTENCE IS . .)
(WHAT IS THE POSITION OF THE PAD Q)

(THE FUNCTION USED IS . .)
LOC-SELECT
((SPECIFIC . PAD))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
LOCATES
(PAD)
(ITS REPLY . .)


(THE LEFT-TO-RIGHT ORDER IS AS FOLLOWS)
(PENCIL (ASH-TRAY BOOK TELEPHONE PAD) PAPER)

(THE NEXT SENTENCE IS . .)
(A TELEPHONE IS AN AUDIO-TRANSDUCER)

(THE FUNCTION USED IS . .)
SETR-SELECT
((GENERIC . TELEPHONE) (GENERIC . AUDIO-TRANSDUCER))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETR
(TELEPHONE AUDIO-TRANSDUCER)
(ITS REPLY . .)
(I UNDERSTAND THE SUPERSET RELATION BETWEEN AUDIO-TRANSDUCER AND TELEPHONE)
(I UNDERSTAND THE SUBSET RELATION BETWEEN TELEPHONE AND AUDIO-TRANSDUCER)

(THE NEXT SENTENCE IS . .)
(A DIAPHRAGM IS PART OF AN AUDIO-TRANSDUCER)

(THE FUNCTION USED IS . .)
PARTR-SELECT
((GENERIC . DIAPHRAGM) (GENERIC . AUDIO-TRANSDUCER))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
PARTR
(DIAPHRAGM AUDIO-TRANSDUCER)
(ITS REPLY . .)
(I UNDERSTAND THE SUBPART-OF-EACH RELATION BETWEEN DIAPHRAGM AND AUDIO-TRANSDUCER)
(I UNDERSTAND THE SUPERPART-OF-EACH RELATION BETWEEN AUDIO-TRANSDUCER AND DIAPHRAGM)

(THE NEXT SENTENCE IS . .)
(WHERE IS A DIAPHRAGM Q)

(THE FUNCTION USED IS . .)
WHERE-SELECT
((GENERIC . DIAPHRAGM))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
WHEREG
(DIAPHRAGM)
(ITS REPLY . .)
(JUST TO THE LEFT OF THE PAD)
(JUST TO THE RIGHT OF THE BOOK)
(SOMEWHERE TO THE LEFT OF THE FOLLOWING . . (PAPER))
```

This empty page was substituted for a blank page in the original document.

Biographical Note

Bertram Raphael was born in New York City on November 16, 1936. He attended the Bronx High School of Science, received a B.S. degree in Physics from Rensselaer Polytechnic Institute in 1957, and received an M.S. degree in Applied Mathematics from Brown University in 1959.

Mr. Raphael held several scholarships at RPI from 1953 to 1957, and the Universal Match Foundation fellowship at Brown University in 1958. He received an NSF honorable mention and was elected to the Society of Sigma Xi in 1957.

Mr. Raphael has been interested in automatic computation since 1959 and has worked in that field for RCA, Moorestown, New Jersey; for Bolt, Beranek and Newman, Inc., Cambridge, Massachusetts; and for the RAND Corporation, Santa Monica, California, for whom he is presently a consultant. He taught at RAND summer institutes for Heuristic Programming (1962) and Simulation of Cognitive Processes (1963), and lectured at UCLA during the summers of 1963 and 1964. He has recently accepted an appointment as Assistant Research Scientist at the Center for Research in Management Science, University of California at Berkeley, effective June, 1964.

His publications include:

"Multiple Scattering of Elastic Waves Involving Mode Conversion," with R. Truell, AFOSR TN 59-399, Metals Research Laboratory, Brown University, May, 1959.

"A Computer Representation for Semantic Information," paper presented at 1963 meeting of AMTCL, abstract in _Mechanical Translation_ _7_ (2), October, 1963.

"A Comparison of List-Processing Computer Languages," with D. G. Bobrow, _Comm. ACM_, expected publication May, 1964.

"LISP as the Language for an Incremental Computer," with L. Lombardi, in _The LISP Programming Language: Its Operation and Applications_, (E. C. Berkeley, ed.), Information International, Maynard, Massachusetts, expected publication May, 1964.

His hobbies include mountain climbing and square dance calling.

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

This empty page was substituted for a blank page in the original document.