The use of theorem-proving techniques in question-answering systems

by C. CORDELL GREEN and
BERTRAM RAPHAEL

Stanford Research Institute
Menlo Park, California

INTRODUCTION

For the purpose of this paper, a question-answering system is a computer program that has at least the following three characteristics:

(1) The ability to accept statements of fact and store them in its memory

(2) The ability to search stored information efficiently and to recognize items that are relevant to a particular query

(3) The ability to respond appropriately to a question by identifying and presenting the answer if it is present in memory, and by deducing a reasonable logical response from relevant knowledge if the complete answer is not explictly available.

The logical deductive ability mentioned in characteristic (3) is a crucial ingredient of any question answerer. In fact, this is the key factor that distinguishes question-answering systems from conventional fact-retrieval systems (such as those used for airline reservations).

This paper describes two programs developed as part of a continuing effort to build more effective question-answering systems. These programs apply predicate calculus theorem-proving methods to question answering. Although research along these lines is by no means complete, the work reported here establishes the feasibility of adapting formal mathematical techniques for use in this important area of artificial intelligence.

PREVIOUS WORK IN QUESTION-ANSWERING SYSTEM

Early work in the general area of question-answering systems is competently surveyed by Simmons.[1] Additional and more recent work has been reported by several groups including Colby and Enea,[2] Craig et al.,[3] Levien and Maron,[4] McCarthy,[5] Quillian,[6] Simmons,[7] Slagle,[8] Thompson,[9] and Weizenbaum.[10]

Some of the previous work, including that of Weizenbaum and Colby and Enea, has been primarily concerned with the form of the input language to the question answerer. These groups are developing ways for a question answerer to accept subsets of natural English as the input language. Others, including Coles[11] and Darlington,[12] have isolated and attacked the problem of translating from English into a formal intermediate language, usually the first-order predicate calculus, that might then be used as input to a question answerer. McCarthy has been largely concerned with the interesting question of how to formalize everyday knowledge. McCarthy's "advice taker" discussions[13] and

Slagle's DEDUCOM[8] system, among others, have proposed the application of formal logical techniques to question answering.

Simmons, Raphael,[14] Levien and Maron, Thompson, and Craig et al. seem to be converging toward the viewpoint that some sort of list structure representation of relational data is necessary in a question-answering system as a general semantic model. Quillian has implemented an elaborately cross-referenced dictionary. This model of English semantic relations should be viewed as a potentially useful tool for (rather than an example of) a question-answering system.

Our current work is based upon both the semantic question-answering work reported by Raphael,[14,15] and the formal theorem-proving techniques developed by Robinson[16] and others. We have constructed two question-answering systems that use list-structured data bases and formal theorem-proving techniques to store facts, extract relevant data, and deduce logical answers to questions. These two systems were programmed in the LISP 1.5 language and implemented via remote terminals on the Q-32 time-shared computer at the System Development Corporation.

KEY PROBLEMS IN QUESTION ANSWERING

Previous work in question answering has pointed up two key problems that must be solved before practical question-answering systems can be developed: the problem of identifying relevant facts and the problem of deducing answers that are not explicit in the data.

Identification of Relevance

The problem of identifying those stored items that are relevant to a particular query does not arise in many experimental systems because their sample data are small enough to be scanned exhaustively. However, this problem of relevance could easily become overwhelming in more realistic situations. Consider, for example, the monkey-and-bananas problem:* Slagle's formalization[8] includes as an axiom, "Under the bananas is a place." Now, suppose the system "knew" that any of an infinity of geographic locations may be considered "places"? It would have to be able to select the relevant place to form the necessary axiom. Similarly, Newell[17] has pointed out that, in most proposed formalizations, as soon as the monkey climbs the box he is stuck on top of it because the axioms include "The monkey can climb onto the box" but not "The monkey can climb off the box." The importance of selectivity becomes obvious when we realize that a knowledgeable system would have not only both these facts to consider as potential axioms, but also "The monkey can jump up and down," "The monkey can scratch his head," "The monkey can go to sleep," etc.

Generality of Inference Deduction

A question-answering system must be able to deduce facts that are not explicitly available in its data base; in other words, it must be capable of logical reasoning. In many previous systems, such logical ability has been introduced in a haphazard manner, if at all. In Raphael's SIR system,[15] for example, a different responding subroutine was used for each of a catalog of question types. Each subroutine had, built into its code, "knowledge" of the logical relations that affected answers to questions of the type that it handled. Although this approach was effective for small experiments, it caused the system to be extremely difficult to expand and modify. New question types and data

*This problem was suggested by McCarthy as a test for question-answering or problem-solving systems. Briefly, one presents as data a description of a situation in which a monkey is located in a room that has a bunch of bananas hanging from the ceiling out of his reach and a box sitting in a corner. Upon receiving the question, "How can the monkey get a banana?" the "intelligent" system would reply, "He pushes the box under the bananas, climbs the box, and then reaches a banana."

structures frequently interacted with previous ones in subtle ways that required major portions of the system to be reprogrammed.

A valuable feature missing from such experimental question answerers is the ability to store in the data base general facts about the relations of the logical system, e.g., the fact that set inclusion is transitive, as well as specific facts about the objects in the real domain, e.g., the fact that John is a member of the set of all boys. If such general information were present in the data, the logical portion of thequestion-answering program could be an isolated theorem-proving routine. Such a routine would permit the experimenter to change both the individual relations and their logical and interactive properties without reprogramming.

In the remainder of this paper we describe two new experimental question-answering systems that have this kind of logical capability. The first, QA1, uses a simple theorem prover that operates upon an elaborate list-structured data base containing both specific facts and logica'l axioms. The second, QA2, represents a major departure from previous approaches to question answering. Here a formal theorem-proving system lies at the heart of the question answerer. The particular structure of the data base is unimportant, at least in our current early version of the system.

QA1 – A THEOREM-PROVING QUESTION ANSWERER BASED ON LIST-STRUCTURED MEMORY

Summary of Approach

QA1 is based largely on the SIR system of Raphael. The major advantage of QA1 over SIR lies in the ability of QA1 to hold in its list-structure memory logical statements about how various kinds of facts might interact. Thus QA1 does not require many separate ad hoc question-answering routines as did SIR.

A modified version of the first-order predicate calculus was selected to represent information from a few of the subject areas that had been used by previous question-answering systems. Statements in this formal language were placed, by the system, on property lists of various key words. Facts were retrieved by searching these property lists, and logical conclusions were deduced by using modus ponens and substitution as rules of inference.

QA1 is the first of what we expect to be a series of successively more powerful question answerers; it was never intended to be a complete polished system. Thus sample problem domains were borrowed from previous question-answering programs, and the control language and logical deduction programs of QA1 were left in rather rough form.

Abilities of QA1

QA1 is capable of accepting facts from any subject domain, provided the input statement is properly posed, and then answering questions based on those facts. The following are examples of questions that the system is capable of answering, after the given facts have been stated in an appropriate input language:

(1) **Part-whole relationships**

   (a) **Facts:** Every person has two arms, and every arm has one hand, and every hand has five fingers.

(2) **Set-membership**

   (a) **Facts:** John is an instance of a person.

   (b) **Questions:** Is John a person? John is an instance of what? Who is an instance of a person?

(3) **Set-inclusion**

   (a) **Facts:** Every boy is a person.

   (b) **Questions:** Is every boy a person? If John is a boy, who is an instance of a person?

(4) **Spatial-relationships**

   (a) **Facts:** The telephone is on top of the table.

   (b) **Questions:** What is on top of the table? If any telephone can be used for communication, what is on top of the table that can be used for communication?

(5) **Miscellaneous**

   (a) **Facts:** Scott is an airfield having an asphalt runway which is 6000 feet long. Scott, Clark, and Dover are all airfields.

   (b) **Question:** What is Clark? Does Scott have a runway? Does Scott have a runway length of of 2000 feet?

Data Representation

Suppose we wish to store in a computer some items of information, which we shall call "the data," for use by a question-answering system. These data may have as their source English text, logical rules, pictures, etc. The problem of representing this data may be divided into three parts:

(1) Determining the semantic content of the data. For example, we may decide that the semantics of the sentence, "John is the father of Bill," is expressed by the binary relation "is-the-father-of" applied to the objects named "John" and "Bill."

(2) Choosing a language in which to express this semaritic content. For example, we may represent the above relation by the following symbols: Father(John, Bill).

(3) Choosing a memory organization–i.e., a way to represent statements in the computer memory. For example, this could consist of a simple string of characters, or of an elaborate cross-referenced list structure.

Part (1) of the representation task, semantic analysis, must be performed by a preprocessor such as a naturallanguage interpretation program or a picture.scanner and interpreter. QA1 and QA2 both assume that this difficult task has already been done.[†]

The first-order predicate calculus was chosen as our language for expressing semantic content, because it is well defined and well understood by mathematicians and also because it resembles the functional notation of LISP 1.5, our principal programming language. The actual input to our programs consists of relational statements in this notation of logic.

The data are represented in QA1 by list structures organized for efficient information retrieval. (See references[19] for a discussion of list structures and property lists.) Binary relations are stored in memory as attribute-value pairs on the property lists of the symbols that are related. Statements of greater complexity than binary relations are also

[†]A question-answering system with considerable background "knowledge" would be a useful component of the semantic analysis system itself, e.g., to help resolve syntactic ambiguities.

stored on these property lists under special attributes. A sample of the data base may be represented as follows:

JOHN → TYPE(proper name), INST(John, Boy)

PERSON → TYPE(generic name), INCLUDE(Boy, Person),
HAS-PART-SET(person, arm, 2)

TELEPHONE → TYPE(generic name), INST(telephone_1, telephone),
USE(telephone, communication)

TELEPHONE_1 → TYPE(proper name), INST(telephone_1, telephone),
ONTOP(telephone_1, table_1)

TABLE → TYPE(generic name), INST(table_1, table)

TABLE_1 → TYPE(proper name), INST(table_1, table),
ONTOP(telephone_1, table_1)

USE → TYPE(relations), ARGUMENTS(2)

INST → TYPE(relation), COMPUTES(INST (ST X RULE(FREEVARS
(U V W) PRED(IMPLIES (AND (INCLUDE V W) (INST U V))
(INST U W)))))

(The word at the tail of each arrow is an elementary symbol. The list at the head of an arrow is the property list of that symbol. Each pair of elements of a property list associated with an elementary symbol consists of the name of an attribute of that symbol, denoted above by an upper-case word, and the value of that attribute for that symbol, denoted above by an expression or list of expressions enclosed in parentheses. INST is the name of the set-membership relation, and INCLUDE is the name of the set-inclusion relation.)

These statements represent the information that: "JOHN" is the name of an object belonging to the set of Boys; "PERSON" is the name of a set that includes the set of Boys; every "PERSON" has as parts a set of two arms; "Telephone_1" is a member of the set of all telephones; all telephones are used for communication; "Table_1" is a member of the set of all tables; "Telephone_1" is on top of "Table_1;" "USE" is a binary relation; "INST" is a relation; and the following axiom holds:

$$(\forall u)(\forall v)(\forall w)[v \subset w \wedge u \in e \Rightarrow u \in w].$$

(This axiom states the following elementary set-theoretic fact: For all u, v, and w, if v is a subset of w and if u is an element of v, then u is an element of w.)

Query Language

To interrogate QA1, the user types questions in one of two formats, depending upon whether the question is to be answered by the FINDR function or the VFIND function.

1. FINDR

   The function FINDR takes as its arguments the name of a relation and the ar-

guments for that relation. Each argument for a relation, called a term, may be completely specified by name, partially specified by a descriptive operator, or unspecified (a "don't-care" term). FINDR will attempt to find objects that satisfy the specifications for each term and for which the relation holds. If it succeeds, it will return the relation with the names of the appropriate objects filled in. Although FINDR has a limited degree of logical inference ability embedded in its specification-matching facility, it is primarily a memory-searching function. It also knows about and deals with the abbreviations used in the memory.

Some examples of its operation are given below. In each case the machine's output is given on the line following the input (which starts with FINDR).

FINDR(INST(TELEPHONE$_1$ TELEPHONE))
(INST(TELEPHONE$_1$ TELEPHONE))

Here all terms are objects specified by name. The statement is found true, and FINDR returns the input unchanged.

FINDR(INST(DCE TELEPHONE))
(INST(TELEPHONE$_1$ TELEPHONE))

(DCE) stands for a don't-care expression–i.e., any term that satisfies the above relations.

FINDR(ONTOP((ST X USE(X COMMUNICATION)) TABLE$_1$))
(ONTOP(TELEPHONE$_1$ TABLE$_1$))

The first argument of the relation ONTOP is partially specified by using the descriptive operator ST ("such that"). The term (ST X USE(X COMMUNICATION)) means "the object X such that X is used for communication." Some logical inference was necessary to answer this question. Three facts–(a) TELEPHONE$_1$ is a telephone, (b) every telephone is used for communication, and (c) TELEPHONE$_1$ is on top of TABLE$_1$–were used in the course of answering this question.

2. VFIND

The function VFIND represents our first attempt at a procedure for applying special axioms found in the data. The arguments of VFIND are a list of variables and a list of predicates containing those variables. The program will then attempt to find values for the variables that will satisfy all of the predicates. VFIND first calls FINDR to look for an explicit answer. IF FINDR fails to find the answer by searching memory, then VFIND searches memory for a relevant axiom and then tries to deduce the answer. If FINDR succeeds, VFIND returns a list of dotted pairs in which each variable is paired with the value that satisfies the predicates. The deductive procedure is described in the section below. Some examples of the use of VFIND are given below.

VFIND((X)(INST JOHN X))
((X . BOY))

VFIND((X)(INST X PERSON))
((X . JOHN))

VFIND((Y)(HASPARTSET HUMAN FINGER Y))
((Y . 10))

This last result may be interpreted precisely to mean that every member of the set of humans has as parts a set of 10 elements from the set of fingers. Each of the above questions required several step deductions and the use of special axioms found in memory.

Deduction Routines

If the answer to a given question was not found in memory, the program VFIND carried out the following procedure: Let us suppose the input is

VFIND((X)(INST X PERSON)),

meaning "find some element in the set of persons." Also, suppose that no information of the form

(INST MIKE PERSON)

is in memory. FINDR will thus fail to find the answer in the first search. It will then search for an axiom and find, on the property list of INST, the rule

(FREEVARS (U V W) PRED(IMPLIES
(AND (INCLUDE V W) (INST U V))
(INST U W))))

that was described in Sec. C above.

Having found an axiom that is probably relevant, VFIND then uses a backward-chaining heuristic similar to that of the Logic Theory Machine[19] in an attempt to deduce an answer. The consequent (INST U W) is "matched" to (INST X PERSON); and then, since the match succeeds, two subproblems consisting of the two predicates in the antecedent

(INCLUDE V PERSON) and

(INST JOHN V)

are generated. Two predicates that match these requirements,

(INCLUDE BOY PERSON) and

(INST JOHN BOY),

are then found in memory. The program keeps track of variable assignments resulting from successful matches and returns the value

((X . JOHN))

The process is recursive, and at any point in a subproblem it may search for additional special axioms. In QA1 all axioms are in the form of implications, so no additional rules of inference are used.

Evaluation and Limitations of QA1

The system was fast, taking only a few seconds of real time for the most difficult questions that it was capable of answering. Exact machine times are unknown because of the difficulty of obtaining accurate timing in the Q-32's time-sharing environment. Once the program was operative, the ability to deal with new subject areas could be added in

a few minutes by merely typing in the necessary relations and axioms. Also, new and old subjects could be interactive. For example, the program would automatically employ set-membership information, if necessary, in solving spatial-relationship problems.

However, QA1's logical deductive ability was drastically incomplete. The program handled existential quantifiers only in certain cases and recognized only two logical connectives, AND and IMPLIES. The functions VFIND and FINDR were not quite compatible; and, as a result, the rule of inference

$$(\forall x)P(x) \Rightarrow P(a)$$

(which says, if a predicate is true for all values of its argument, then it must be true for a particular value of its argument) could not be applied in some cases. The program had no sophisticated means of preventing loops or picking the order in which to attempt deductions. It tried a simple depth-first search[‡] of the proof tree generated by the deductive routine described in Sec. E above. As a result of these limitations, QA1 could not answer such questions as "How many hands does John have?" and "Does there exist a person who is not a boy?" The formalization of these two problems and their solutions by QA2 are given in a section below.

To progress further, there were two alternatives:

(1) To modify the current program by correcting each deficiency one at a time and experimentally evolve more sophisticated deductive routines, perhaps similar to those of Fischer Black's questionanswering program.[20]

(2) To base our new work upon relevant research in the field of automatic theorem proving.

Like question-answering programs, effective automatic theorem-proving programs must be logically complete and must contain heuristics for selecting subproblems–i.e., for searching "proof trees" efficiently. To our knowledge, however, theorem provers have not yet been used in systems containing information-retrieval capabilities. It was not clear just how a mathematical theorem prover could be used.

We selected the second alternative-adaptation of results in automatic theorem proving–because of its potential power to provide us eventually with a very general, yet conceptually simple, question-answering system. Thus work on QA1 was abandoned and we proceeded to study how results from mathematical theorem proving could best be utilized, and then to implement QA2.

THEOREM PROVING AND QUESTION ANSWERING

One of the most important characteristics of a question-answering system is its logical deductive ability. A system that can derive new responses from its stored knowledge is far more interesting than a system that can only parrot back responses that are stored explictly in its memory.

Mathematicians and philosophers have studied the nature of implication and deduction, primarily in the abstract domain of formal logic, Most of the formal logical systems that have been studied contain all the reasonable properties one might wish in "deducing" facts in a particular, informal subject domain, and most formal systems can be easily applied, with appropriate semantic models, to the particular subject domains of interest. The system QA2 attempts to apply the most powerful logical procedures reported in the mathematics literature to our question-answering problems.

[‡]A "depth-first search" is one which attempts such subproblem as soon as it arises, in contrast with a "breadth-first search" which defines all subproblems at each level before attempting to solve the first at the next level.

Review of Formal Theorem Proving[§]

Formal logic usually deals with well-defined strings of symbols called "well-formed formulas" (wff's), and with a subset of the wff's called "theorems." Each wff can be interpreted as a statement, which may be true or false, about the state of a particular semantic model. The semantic domain may consist of any objects and relations; in the absence of specific semantic knowledge, a domain consisting of numbers and sets is frequently used as the "standard interpretation."

A model is said to satisfy a wff if the statement represented by the wff is true for that model. A wff that is satisfied by all possible models (from the semantic domain) is called valid.

The theorems of a logical system are usually intended to be the valid wff's. However, since it is not practical in general to enumerate and test all possible models, formal syntactic procedures called proof procedures must be used to establish theorems. If every theorem of a proof procedure is indeed valid, the procedure is called sound. If every valid formula can be demonstrated to be a theorem, the procedure is complete. In the desirable case that a proof procedure is both sound and complete, the theorems of the procedure coincide with the valid wff's. A decision procedure is a sound and complete proof procedure that can effectively decide whether any given wff is valid or not.

Unfortunately, a famous theorem by Gödel shows that any consistent and sufficiently rich formal system is incomplete; that is, wff's will always exist that are valid but that cannot be formally proved to be valid. This means that, for the interesting formal systems, there can be no decision procedure; we must content ourselves with sound proof procedures that can establish as theorems some, but not all, of the valid wff's.

As a practical matter, however, the incompleteness property is much less restrictive than it may at first appear. Because of the time and space constraints on practical computation, the heuristic power of a proof procedure–i.e., its ability to prove useful theorems efficiently–is more important than its ultimate effectiveness on all theorems. A decision procedure that requires enormous amounts of time or intermediate storage for some wff's is indistinguishable, in practice, from an incomplete proof procedure that never terminates for those wff's.

In recent years, much work has been done on the development of proof procedures suitable for implementation on a ditigal computer. The most effective of these seem to be those that use the Robinson resolution principle[16] of deduction for the area of logic called "first-order predicate calculus."

The Robinson Procedure for Proof by Resolution

One approach to theorem proving in the first-order predicate calculus, due to Herbrand, is to attempt to specify the set of models that satisfies the negation of the wff to be proved. Since every wff is either true or false for each possible model, every model must satisfy either a given wff or its negation, but not both. Only three results can occur from the attempted specification of a set of models satisfying the negation of a wff:

(1) The specification process leads to obviously contradictory model assignments, proving that no satisfying model is possible and the wff is valid.

(2) The specification process terminates without a contradiction, proving that a satisfying model exists for the negation so that the wff is not valid.

(3) The process does not terminate in either state (1) or (2) in a finite amount of time,

[§]This subsection and the next are included to establish the background and terminology for our discussion of QA2. Readers interested in more complete presentations of these topics should see references 21, 22, 23.

and the validity of the wff is still undetermined.

This is why such proof procedures are not decision procedures.

The Robinson proof procedure uses the resolution principle in conjunction with the Herbrand approach to theorem proving. The negation of the wff to be proved is first placed into a standard form.¶ In this form, the wff is represented as the conjunction of a set of formulas called clauses, each of which is a disjunction of elementary formulas called literals. Then new clauses are deduced from the starting clauses by the inference rule of resolution, described below, such that the original wff is satisfiable only if its descendent clauses are all satisfiable. The goal of the procedure is to deduce a contradictory formula of the form $\alpha \wedge \sim\alpha$ which is clearly not satisfiable and therefore demonstrates that the starting wff is not satisfiable.

The rule of resolution is best illustrated first in its propositional form: if $p \vee \alpha$ and $\sim p \vee \beta$ are two wff's in which p is any proposition and α and β are any wff's, one may deduce the wff $\alpha \vee \beta$.

The predicate calculus form of the resolution rule is this: Let L_1 be any atomic formula–i.e., a wff consisting of a single predicate symbol followed by an appropriate set of terms (constants, variables, or functions) for arguments. Let L_2 be the negation of an atomic formula consisting of the same predicate symbol as L_1 but generally with different arguments. Let α and β be any wff's in the predicate calculus. Let $(\alpha)\sigma$ be the wff obtained from α by making all substitutions specified by the substitution set σ, of formulas for free occurrences of variables in α. If any set of substitutions σ_1 for variables in L_1 and L_2 makes L_2 identical to the negation of L_1, then from the two wff's $L_1 \vee \alpha$ and $L_2 \vee \beta$ we may deduce the "resolvent" $(\alpha \vee \beta)_\sigma$.

Example:

$$P(x, f(y)) \vee Q(x) \vee R(f(a), y)$$

and

$$\sim P(f(f(a)), z) \vee R(z, w)$$

imply, by resolution,

$$Q(f(f(a))) \vee R(f(a), y) \vee R(f(y), w)$$

where the substitution set is $\sigma = \{(f(f(a)) \text{ for } x, f(y) \text{ for } z\}$.

The main theorem of resolution states that if a resolvent is not satisfiable then neither is the conjunction of its antecedents, and that the empty formula, obtained by resolving two clauses of the form L and $\sim L$, is not satisfiable.

The resolution rule tells us how to derive a new clause from a specified pair of clauses containing a specified literal, but it does not tell us how to choose which clauses to resolve. A mechanical attempt to resolve all possible pairs of clauses generally results in the generation of an unmanageably large number of irrelevant clauses. Therefore, various heuristic search principles are being developed to guide and control the selection of clauses for resolution. These heuristics, called set of support, unit preference, level bound, subsumption, etc., are described in the literature.[16,22,25,26,27]

Utilizing a Theorem Prover

The theorem-proving techniques described above have been developed for the purpose of proving theorems in mathematics. Let us consider how these results may be applied

¶For those familiar with Mathematical Logic, this form is prenex conjunctive normal form, in which existentially quantified variables are replaced by Skolem functions[24] of appropriate universally quantified variables.

to the question-answering problem:

(1) Interesting mathematical theorems usually require lengthy, complex proofs. The logic of everyday reasoning, on the other hand, is relatively simple, once the appropriate premises have been identified; although people commonly draw upon a wide range of knowledge for their premises, they rarely perform intricate chains of deductive steps such as those needed to prove a theorem in Group Theory. Thus one can expect the theorem-proving techniques currently being developed by mathematicians to be useful in question answering even before they are powerful enough to be valuable answering even before they are powerful enough to be valuable in mathematics. Of course, each improvement made in mathematical theorem provers can be carried over into a corresponding improvement in the effectiveness of a theorem-proving question answerer.

(2) The fact that the theorem prover is a well-defined subroutine independent of the subject matter or memory organization of its data is important for the generality of the question answerer.

(3) Any "true or false" question can be stated in a natural way as a theorem to be proved. To find out if a given input sentence is true or false the theorem prover will attempt first to prove that the sentence is true. If, after a certain expenditure of effort, no proof is found, the theorem prover could then attempt to prove the sentence false.

(4) A question that asks for a specific item of information can usually be stated in the form, "Find x such that $P(x)$ is true," where $P(x)$ is some specified predicate. This problem may be posed to a theorem prover as the statement $(\exists x)(Px)$. If this statement is proved, then the answer to the question is the term that is substituted for x during the course of the proof. This term may be a variable (signifying that $P(x)$ is true for all x), a constant, or a function. If the clause representing $P(x)$ is used several times in the proof, then the answer is the disjunction of the several terms substituted for x. This answer may then be represented internally as a clause, e.g., $\{P(a) \vee P(b) \vee P(x) \vee P(f(y))\}$ so that the theorem prover may then simplify it by removing unnecessary disjuncts.

In the next section we describe and give examples of the operation of a question-answering system, QA2, that uses a theorem prover in just these ways.

QA2–AN EXTENDED THEOREM PROVER AS A QUESTION ANSWERER

Summary of Approach

QA2 is a question-answering system derived almost exclusively from formal theorem-proving ideas. It is based upon the Robinson proof procedure described in Sec. V-B and uses the "unit preference strategy"[25] and other heuristics to improve efficiency of proof generation.

The input to QA2 consists of relational statements in the same standard logical notation that was used for QA1. However, instead of storing these statements on the property lists of key words, QA2 transforms each statement into the standard conjunctive form needed by the Robinson theorem-proving technique. Then it stores the resulting set of clauses in a simple list structure indexed by relational symbols.

The problem of finding which facts are "relevant" to a query (or a theorem) is equivalent to the problem of deciding which fact to try to use next in the process of answering the query. In the context of QA2, this becomes the problem of which clause to use next

in the process of proving the theorem. A simple and logically complete solution to this problem is given by the following extension of the set-of-support strategy:[25]

(1) First, the theorem prover is given only the clauses representing the negation of the sentence to be proved. All clauses representing this negated sentence are said to be in the "set-of-support." Then an attempt is made to construct a proof. (Note that a theorem of the predicate calculus–e.g., $(\forall x)[P(x) \vee \sim P(x)]$–may be provable without reference to facts in memory.)

(2) If no proof is found, the theorem prover then addresses memory for a limited number of additional clauses that will resolve with clauses in the theorem prover's set-of-support; i.e., additional clauses from which new clauses can be deduced by the resolution method. (Suitable memory organization and use of the "subsumption"[16] heuristic can be used to increase the efficiency of the search.)

(3) If no proof is found with the new clauses, then Step 2 is repeated to obtain more "relevant" facts from memory.

As in other theorem-proving programs, heuristics such as a bound on level or computing time must be used to ensure practical run times for the program.

The measure of "relevance" of one clause to another is, then, whether or not the clauses will resolve. Note that this process is complete in the sense that if a proof exists (within the limitation on time and space) it will be found. The process is efficient in the sense that some clauses that cannot lead to a proof are never used.

QA2, the program described below, contains an implementation of the above algorithm. Although this technique has the advantage of relative efficiency over a more direct approach like, say, giving all the clauses in memory to the theorem prover, several further improvements will be necessary before we produce a truly practical system.

QA2–Control Language

The question-answering program for QA2 consists of a collection of functions or subprograms that perform the various tasks necessary for such a system. At the top level an executive program EXEC allows for user-machine interaction by accepting input in its "command language," calling upon the appropriate function to perform the desired operation, and responding to the teletype user. At present, the language accepts three types of input: statements, questions, and commands.

1. **Statements**

   A statement is entered in the following format:

   S expression

   where the letter S signifies that the following "expression" is to be added to the system's data base

   The expression is a predicate calculus statements such as

   (IN JOHN BOY)

   or

   ((FA (X Y Z) (IF (AND (IN X Y) (INCLUDE Y Z)) (IN X Z))).

   The first states that John is a boy, or more precisely, that John is an element of the set named Boy.

The second is equivalent to the predicate calculus statement:

$$(\forall x)(\forall y)(\forall z)[x \in y \wedge y \subset z \Rightarrow x \in z]$$

2. **Questions**

   A question is entered in a similar fashion:

   Q question

   where Q signifies that the predicate calculus expression that follows is to be treated as a question to the system. Here, the negation of the question is put into the appropriate standard form and passed on to a subexecutive program EXEC1, which attempts to answer the question based on the current information in the data base. (Examples of QA2 in operation shows how various questions may be posed as predicate calculus expressions.)

3. **Commands**

   The following series of additional commands have been implemented, allowing the user to interrogate and alter the system:

   (1) UNWIND
   After a question has been successfully answered, the UNWIND command will print the proof of the answer given to the question.

   (2) CONTINUE
   If the system was unsuccessful in answering a question, the CONTINUE command will cause the system to continue searching for proof with the level bound raised.

   (3) LIST
   The command LIST PR where PR is a predicate symbol will list all of the statements in the data base that contain the symbol PR.

   (4) FORGET
   The command FORGET PR S will delete certain statements that contain the predicate letter PR according to the format of S–e.g., if S is an integer n, the n^{th} statement will be deleted.

   (5) FILE
   FILE F asks the theorem prover to operate on a prepared list F of clauses.

The Theorem Prover: Technical Considerations

The theorem prover accepts as input a list of clauses CLAUSELIST and a level bound MAXLEV. Its goal is to determine, if possible, that the set of clauses is unsatisfiable, or equivalently, to derive the null clause (containing no literals) that results from two contradictory clauses.

The algorithm uses the unit preference strategy with set-of-support. In addition, it traces the values assigned to variables contained in the question that were originally bound by existential quantifiers. Thus if the theorem prover completes a proof of a statement of the form $(\exists x)P(x)$, the question answerer can frequently exhibit the x that satisfies P. This is extremely useful, as the examples in "Examples of QA2 in Operation" below will show.

The operation of the theorem prover starts by ordering the clauses on CLAUSELIST by the number of literals in each clause. The program successively attempts to produce resolvents from the clauses in CLAUSELIST, producing first those resolvents of shortest length. To avoid redundant computation as much as possible, resolvents of two clauses C1 and C2 are produced only if the following criteria are satisfied:

(1) Either C1 or C2 (or both) are in the set-of-support

(2) The level ℓ of any resolvent of C1 and C2 plus the length of the resolvent must not be greater than the level bound MAXLEV. (This is a modification of the usual level bound strategy.)

(3) Neither C1 nor C2 has been "subsumed" by any other clause in the proof.

Furthermore, if a resolvent R of C1 and C2 is produced, it is added to CLAUSELIST only if R is not a tautology–i.e., does not contain complementary literals–and if R is not subsumed by any clause already on the list.

Examples of QA2 in Operation

A sample dialogue with QA2 is given below. The input and output from the computer are printed in all capital letters. After some of the exchanges, we have added an explanation.

S (IN JOHN BOY)
OK

The statement (indicated by "S") that John is contained in the set of boys is accepted and the response is "OK."

Q (IN JOHN BOY)
YES

The question (indicated by "Q"), "Is John in the set of boys?" is answered "Yes." This is an example of a simple yes or "no proof found" answer.

Q (EX(X) (IN JOHN X))
YES WHEN X = BOY

Does an x exist such that John is in the set x? Note that the program reports what assignment is made to x to complete its proof.

S (FA(X) (IF (IN X BOY) (IN X PERSON)))
OK

This says that every boy is a person, or $(\forall x)[x \in BOY \Rightarrow x \in PERSON]$

Q (EX(X) (IN X PERSON))
YES WHEN X = JOHN

Does a member of the set of all persons exist? The theorem prover must have used two statements: John is a boy, and every boy is a person.

```
UNWIND
   SUMMARY
   1 IN(JOHN,BOY)                AXIOM
   2 −IN(X,PERSON)               NEG OF THM
   3 −IN(X,BOY)
      IN(X,PERSON)               AXIOM
   4 −IN(X,BOY)                  FROM 2,3
   (CONTRADICTION FROM CLAUSES 1 AND 4)
   (5 CLAUSES GENERATED)
```

The command UNWIND caused the proof to be printed. Each numbered line corresponds to one clause. Each clause is the disjunction (logical "or") of the literals it contains. In this example, all clauses contain only one literal except clause 3 which contains two literals. (Clause 3 may be read, "either x is not a boy or x is a person.") A clause may come from three sources:

AXIOM	– retrieved from memory
NEG OF THM	– the negation of the question
FROM N,M	– the result of resolving together clauses N and M.

The number of clauses generated represents the size of the proof tree upon generating the empty clause; this is a measure of the amount of effort involved in completing the proof.

```
S (FA(X) (IF (IN X PERSON) (IN X HUMAN)))
  OK
```

It unquestioningly believes that all persons are human.

```
Q (EX(X) (IN X HUMAN))
   YES WHEN X = JOHN
 S (FA(X) (IF (IN X HUMAN) (HP X ARM 2)))
   OK
Q (HP JOHN ARM 2)
   YES
```

(HP JOHN ARM 2) means that John has-as-parts two elements of the set of all arms.

```
S (FA(Y) (IF (IN Y ARM) (HP Y HAND 1)))
  OK
Q (EX(X) (HP JOHN HAND X))
  NO PROOF FOUND
```

The crucial axiom, given next, was missing.

```
S (FA(X Y Z M N) (IF (AND (HP X Y M)
  (FA(U) (IF (IN U Y) (HP U Z N))) )
  (HP X Z (TIMES M N))) )
  OK
Q (EX(N) (HP JOHN HAND Y))
  YES WHEN Y = TIMES(2, 1)
```

TIMES(2,1) represents the product of 2 and 1 (=2).

```
UNWIND
  SUMMARY
  1  IN(JOHN,BOY)                                  AXIOM
  2  -HP(JOHN,HAND,N)                              NEG OF THM
  3   IN(SK8(N,M,Z,Y,X),Y)
     -HP(X,Y,M)
      HP(X,Z,TIM ES(M ,N))                         AXIOM
  4  -HP(JOHN,Y,M)
      IN(SK8(N,M,HAND,Y,JOHN),Y)                   FROM 2, 3
  5  -IN(Y,ARM)
      HP(Y,HAND,1)                                 AXIOM
  6  -HP(JOHN,ARM,M)
      HP(SK8(N,M,HAND,ARM,JOHN),HAND,1)            FROM 4, 5
  7  -HP(SK8(N,M,Z,Y,X),Z,N)
     -HP(X,Y,M)
      HP(X,Z,TIMES(M,N))                           AXIOM
  8  -HP(JOHN,Y,M)
     -HP(SK8(N,M,HAND,Y,JOHN),HAND,N)              FROM 2, 7
  9  -HP(JOHN,ARM,M) FROM 6,8
  10 -IN(X,HUMAN)
      HP(X,ARM,2)                                  AXIOM
  11 -IN(JOHN,HUMAN)                               FROM 9, 10
  12 -IN(X,PERSON)
      IN(X,HUMAN) AXIOM
  13 -IN(JOHN,PERSON)                              FROM 11, 12
  14 -IN(X,BOY)
      IN(X,PERSON)                                 AXIOM
  15 -IN(JOHN,BOY)                                 FROM 13, 14
  (CONTRADICTION FROM CLAUSES 1 and 15)
  (27 CLAUSES GENERATED)
```

This required an eight-step proof. SK8 is the name generated by the program for a Skolem function[24] used to eliminate an existential quantifier. This elimination of quantifiers is part of the process of converting input statements into standard form for use of the theorem prover.

```
S (OR (AT JOHN HOME) (AT JOHN SRI) (AT JOHN AIRPORT))
  OK
Q (EX(X) (AT JOHN X))
  YES WHEN X = SRI
   OR WHEN X = AIRPORT
   OR WHEN X = HOME
```

Here the system could determine that x has one of a specified set of values; it could not determine precisely which one. In this case, any of the three alternatives is possible. In general, the theorem prover narrows down alternative variable assignments just far enough to establish the truth of the theorem. The resulting list might occasionally contain extraneous alternatives.

S (FA(X) (EQUALS X X))
  OK

S (FA(X Y) (IF (EQUALS X Y) (EQUALS Y X)))
  OK

S (FA(X Y Z) (IF (AND (EQUALS X Y)(EQUALS Y Z)) (EQUALS X Z)))
  OK

S (FA(X Y Z) (IF (AND (AT X Y)(AT X Z)) (EQUALS Y Z)))
  OK

S (NOT (EQUALS SRI AIRPORT))
  OK

S (NOT (EQUALS AIRPORT HOME))
  OK

Q (EX(X) (IF (NOT (AT JOHN AIRPORT)) (AT JOHN X)))
   YES WHEN X = HOME
    OR WHEN X = SRI

S (AT JOHN SRI)
  OK

Q (NOT (AT JOHN AIRPORT))
   YES

S (IF (AT JOHN AIRPORT) (WITH JOHN BILL))
  OK

S (FA(X Y Z) (IF (AND (AT X Y)(WITH Z X)) (AT Z Y)))
  OK

Q (EX(X) (IF (AT JOHN AIRPORT) (AT BILL X)))
   NO PROOF FOUND

S (FA(X Y) (IF (WITH X Y) (WITH Y X)))
  OK

Q (EX(X) (IF (AT JOHN AIRPORT) (AT BILL X)))
   YES WHEN X = AIRPORT

S (FA(X Y) (IFF (DISJOINT X Y) (FA(U) (IF (IN U X) (NOT (IN U Y))))))
  OK

Q (FA(X Y) (IF (DISJOINT X Y) (DISJOINT Y X)))
   YES

S (DISJOINT BOY GIRL)
  OK

S (IN JOHN BOY)
  OK

Q (NOT (IN JOHN GIRL))
   YES

S (IN JUDY GIRL)
  OK

S (FA(X Y Z) (IF (AND (IN X Y)(INCLUDE Y Z)) (IN X Z)))
  OK

S (INCLUDE BOY PERSON)
  OK

```
Q (EX(X) (IN X PERSON))
   YES WHEN X = JOHN
S (INCLUDE GIRL PERSON)
  OK
Q (EX(X) (AND (NOT (IN X BOY))(IN X PERSON)))
   YES WHEN X =JUDY


UNWIND
   SUMMARY
   1  DISJOINT(BOY,GIRL)          AXIOM
   2  INCLUDE(GIRL,PERSON)        AXIOM
   3  IN(JUDY,GIRL)               AXIOM
   4  IN(X,BOY)
      -IN(X,PERSON)               NEG OF THM
   5  -INCLUDE(Y,Z)
      -IN(X,Y) IN(X,Z)            AXIOM
   6  IN(X,BOY) -IN(X,Y)
      -INCLUDE(Y,PERSON)          FROM 4, 5
   7  -INCLUDE(GIRL,PERSON)
      IN(JUDY,BOY)                FROM 3, 6
   8  IN(JUDY,BOY)                FROM 2, 7
   9  -DISJOINT(X,Y)
      -IN(U,X)
      -IN(U,Y)                    AXIOM
   l0 -IN(JUDY,Y)
      -DISJOINT(BOY,Y)            FROM 8, 9
   11 -IN(JUDY,GIRL)              FROM 1, 10
   (CONTRADICTION FROM CLAUSES 11 AND 3)
   (92 CLAUSES GENERATED)
```

CONCLUSIONS

Effective automatic question answering requires significant logical deductive ability. The programs QA1 and QA2 described above represent our attempts to pin down this relationship between question answering and logic. Mathematicians such as Robinson have recently made significant progress toward automating formal logical procedures. The QA2 system demonstrates how a question-answering system can take advantage of the new theorem-proving techniques. The organization of QA2 represents a novel departure from previous comparable systems. In most systems (including SIR and QA1) memory organization and data-retrieval procedures were established first, and then logical deduction procedures were "tacked on" almost as an afterthought. For QA2 we first decided to use the most powerful logical tools available, and then structured the memory and the retrieval procedures for the convenience of the theorem prover.

The results have been most promising. The nature of the inference rules in the theorem prover has determined an effective criterion for choosing "relevant" facts from memory. Question-answering power has so far been insensitive to the particular organization of clauses ("facts") in memory. As the amount of information in the data store grows and the complexity of questions increases, we expect to need better criteria of relevance and more elaborately indexed data; however, the nature of the criteria and the index scheme

will have to evolve in conjunction with more effective search heuristics for the theorem prover.

A major advantage of a theorem-proving question answerer is its generality. It can answer questions about any domain while using the same basic deductive program, provided the nature of the domain is suitably described by a set of axioms. Of course, this approach is not as efficient for answering questions about particular domain as an approach based upon special subroutines and models for that domain. We are currently implementing the ability of the theorem prover to regain the efficiency of a special-purpose program by evaluating specified functions that appeal to "outside" information about the semantics of a subject domain when convenient. The ability to evaluate arithmetic functions and predicates without having to derive the answers from an axiomatization of arithmetic is first step in this direction.

The formal mathematical nature of the input to our systems–predicate calculus–is an obstacle to free conversational experimentation by an untrained user. Fortunately, the output from the natural language processing system developed by Coles[11] is almost precisely the appropriate language for input to QA2. We are presently coupling these two systems.

Finally, a major goal for our current work is to apply QA2 to problem solving, in addition to question answering, tasks. If permissable actions are suitably axiomatized, then the steps in a proof of the assertion that a problem's goal is achievable can be used to indicate the sequence of actions that must be taken to achieve it. We have hand-simulated solutions to such diverse problems as writing a computer program and "reaching the bananas," with encouraging preliminary results.

ACKNOWLEDGEMENT

The research reported here was supported at Stanford Research Institute by Contract AF 19(628)-5919 Project 4641, Air Force Cambridge Research Laboratories, Electronic Systems Division, Air Force System Command, USAF, but the report does not necessarily reflect endorsement by the sponsor.

The authors are indebted to Robert A. Yates for his substantial contributions to the design and programming of system QA2.

REFERENCES

1 R F SIMMONS
Answering english questions by computer: a survey
Comm ACM Vol 8 No 1 January 1965

2 K M COLBY H ENEA
Heuristic methods for computer understanding of natural language in context-restricted on-line dialogue
Dept of Computer Sciences Stanford University 196?

3 J A CRAIG et al
DEACON: direct english access and control
AFIPS Proc FJCC Vol 29 1966

4 R E LEVIEN M E MARON
A computer system for inference, execution and data retrieval
Comm ACM Vol 10 No 11 pp 715-721 November 1967

5 J McCARTHY
Situations, actions and casual laws

Memo No 2 Stanford Artificial Intelligence Project
Stanford University July 1963

6 R QUILLIAN
AFIPS Proc SJCC Vol 30 1967

7 R F SIMMONS
An approach toward answering english questions from text
AFIPS Proc FJCC Vol 29 1966

8 J R SLAGLE
Experiments with a deductive Q-A program
Comm ACM Vol 8 No 12 December 1965

9 F B THOMPSON
English for computer
AFIPS Proc FJCC Vol 29 1966

10 J W WEIZENBAUM
ELIZA—a computer program for the study of natural language communication between man and machine
Comm ACM Vol 9 No 1 January 1966

11 L S COLES
An on-line question-answering system with natural language and pictorial input
(Paper to be presented at the ACM Conference August 1968)

12 J L DARLINGTON
Machine methods for improving logical arguments expressed in english
Mechanical Translation Vol 8 Nos 3 and 4 pp 41-47 June and October 1965

13 J McCARTHY
Programs with common sense
Memo No 7 Stanford Artificial Intelligence Project
Stanford University September 1963

14 B RAPHAEL
A computer program which 'understands'
AFIPS Proc FJCC Vol 26 1964

15 B RAPHAEL
SIR: A computer program for semantic information retrieval
MAC-TR2 Project MAC MIT June 1964

16 J A ROBINSON
A machine-oriented logic based on the resolution principle
J ACM Vol 12 No 1 January 1965

17 A NEWELL
Unpublished seminar talk

18 B RAPHAEL
Aspects and applications of symbol manipulation
Proc 1966 National Conference ACM 1966

19 A NEWELL J C SHAW H A SIMON
Empirical explorations of the logic theory machine: a case study in heuristics
Paper presented at the Western Joint Computer Conference
Los Angeles February 28 1957

20 F BLACK
A deductive question-answering system
Harvard University Ph D Thesis 1964

21 D C COOPER
Theorem proving in computers
Advances in Programming and Non-Numerical Computation L FOX ed Pergamon Press 1966

22 J A ROBINSON
A review of automatic theorem-proving
American Mathematical Society Symposia on Applied Mathematics XIX 1967 Rice University (to be published)

23 E MENDELSON
Introduction to mathematical logic
van Nostrand 1964

24 KALISH and MONTAGUE
Logic: techniques of formal reasoning
Harcourt Brace and World 1964

25 L WOS et al
The unit preference strategy in theorem proving
AFIPS Proc FJCC Vol 26 1964

26 T P HART
A useful algebraic property of Robinson's unification algorithm
Memo No 91 AI Project Project MAC MIT 1965

27 J R SLAGLE
Automatic theorem-proving with renameable and semantic resolution
J ACM Vol 14 No 4 October 1967