11

Theorem-Proving by Resolution as a Basis for Question-Answering Systems

Cordell Green
Stanford Research Institute
Menlo Park, California

ABSTRACT

This paper shows how a question-answering system can be constructed using first-order logic as its language and a resolution-type theorem-prover as its deductive mechanism. A working computer-program, QA3, based on these ideas is described. The performance of the program compares favorably with several other general question-answering systems.

1. QUESTION ANSWERING

A question-answering system accepts information about some subject areas and answers questions by utilizing this information. The type of question-answering system considered in this paper is ideally one having the following features:

1. A language general enough to describe any reasonable question-answering subjects and express desired questions and answers.
2. The ability to search efficiently the stored information and recognize items that are relevant to a particular query.
3. The ability to derive an answer that is not stored explicitly, but that is derivable by the use of moderate effort from the stored facts.
4. Interactions between subject areas; for example, if the system has facts about Subject A and Subject B, then it should be able to answer a question that requires the use of both sets of facts.
5. Capability of allowing the user to add new facts or replace old facts conveniently.

This paper argues the case for formal methods to achieve such a system and presents one particular approach in detail. A natural language facility is not one of the properties sought after or discussed (although Coles, 1968, has

added to the program described here a translator from a subset of English to first-order logic).

The name 'question-answering system' requires clarification. The system described above might be named an 'advice taker' or a 'multi-purpose problem-solving system' or 'general problem-solving system'. McCarthy (1958) proposed using formal languages and deduction to construct such a system, and suggested allowing the user to give hints or advice on how to answer a question; he referred to the proposed system as an 'advice taker'. Research on 'multi-purpose' or 'general problem-solving' tends to differ from question-answering as described above by placing more emphasis on solving deeper, more difficult problems and less emphasis on user interaction, formality, and efficient retrieval of relevant facts from a large data base. The situation is further confused by the use of 'question-answering' to refer sometimes to natural language systems, sometimes to information retrieval systems having little deductive ability, and sometimes to systems with deductive ability limited to the propositional calculus.

It is important to emphasize the distinction between general versus special-purpose question-answering. If the class of questions asked of a system is small, completely specified in advance, and concerned with a particular subject area, such as the question-answering system of Green, Wolf, Chomsky, and Laughery (1963) concerned with baseball, or that of Lindsay (1963) concerned with family relations, then we shall call such a system 'special-purpose'. Frequently the goal in designing a special-purpose system is to achieve good performance, measured in terms of running speed and memory utilization. In this case the best approach may be first to construct a special data base or memory that is optimized for that subject area and question class, and then to write special question-answering subroutines that are optimized for the particular data base and question class. On the other hand, a 'general' question-answering system is one that allows arbitrary subject areas, arbitrary questions, and arbitrary interactions between subject areas during the process of answering a question. This paper describes a rather formal approach to designing a general question-answering system. A precise name for our system is 'a general, formal, deductive, question-answering system.'

2. THEOREM-PROVING

The use of a theorem-prover as a question-answerer can be explained very simply. The question-answerer's knowledge of the world is expressed as a set of axioms, and the questions asked it are presented as theorems to be proved. The process of proving the theorem is the process of deducing the answer to the question. For example, the fact 'George is at home', is presented as the axiom, $AT(George, home)$. The question 'Is George at home?' is presented as the conjectured theorem, $AT(George, home)$. If this theorem is proved true, the answer is yes. (In this simple example the theorem is obviously true

since the axiom *is* the theorem.) The theorem-prover can also be used to find or construct an object satisfying some specified conditions. For example, the question 'Where is George?' requires finding the place x satisfying $AT(George, x)$. The theorem-prover is embedded in a system that controls the theorem-prover, manages the data base, and interacts with the user. These ideas are explained in more detail below.

Even though it is clear that theorem-proving can be used for question-answering, why should one use these very formal methods? Theorem-proving may be a good approach to the achievement of generality for several reasons:

1. The language is well defined, unambiguous, and rather general, so that one may hope to describe many desired subjects, questions, or answers.
2. The proof procedure used allows all possible interaction among the axioms and is logically 'complete' that is, if a theorem is a logical consequence of the axioms, then this procedure will find a proof, given enough time and space. This completeness property is important since several general question-answering programs have resulted in incomplete deductive systems, even in the practical sense of being unable to answer some simple types of questions that are short, reasonable deductions from the stored facts – for example, the author's QA1 (Green and Raphael 1968), Raphael's SIR (1964), Slagle's DEDUCOM (1965), and Safier's SIMPLE SIMON (1965). (However, the fact that we use a first-order logic theorem-prover does impose certain important restrictions discussed in section 5.)
3. The theorem-prover is subject-independent, so that to describe a new subject or modify a previous description of a subject, only the axioms need to be changed, and it is not necessary to make any changes in the program.
4. Formal techniques such as those developed here may be generally valuable to the field of artificial intelligence. The use of a formal framework can lead to insights and generalizations that are difficult to develop while working with an *ad hoc* system. A common, well-defined framework facilitates communication between researchers, and helps to unify and relate diverse results that are difficult to compare.
5. Theorem-provers are becoming more efficient. Even though the theorem-proving method used is theoretically complete, in practice its ability to find proofs is limited by the availability of computer time and storage space. However, the method of 'Resolution' (Robinson 1965), used by the program described here, has been developed to the point of having several good heuristics. Further improvements in theorem-proving are very likely, and, hopefully, the improvements will carry over into corresponding improvements in question-answering.

It should be possible to communicate precisely new theorem-proving results to other researchers, and it is relatively easy to communicate precisely particular formalizations or axiomatizations of subjects.

3. EXTENDING THEOREM-PROVING TO QUESTION-ANSWERING

This section describes, in general, how questions can be asked in first-order logic, and how answers can be generated. Examples illustrating these methods are presented. The discussion in this section and the following two assumes that the reader is somewhat familiar with logic and automatic theorem-proving. An introduction to automatic theorem-proving is given in Cooper (1966) and Davis (1963). The theorem-proving methods mentioned in this paper use the Resolution Principle proposed by J. A. Robinson (1965 and 1967). Additional strategies for using the Resolution principle are presented by Wos *et al.* (1964, 1965 and 1967). This last paper defines terms the 'Extended Set of Support' strategy, 'degree', and 'singly connectedness', that are used in section 4.

The explanation of question-answering given in this section will be illustrated primarily by the techniques used in a working question-answering program called QA3. It is programmed in LISP on the SDS 940 computer, operating in the time-sharing mode. The user works at a teletype, entering statements and questions, and receiving replies. The notation in this paper is slightly different from the actual computer input and output, as the character set available on the teletype does not contain the symbols we use here. QA3 is an outgrowth of QA2 (Green and Raphael 1968), an earlier system, but is somewhat more sophisticated and practical, and is now being used for several applications.

1. Types of questions and answers

Facts are presented as statements of first-order logic. The statement is preceded by STATEMENT to indicate to the program that it is a statement. These statements (axioms) are automatically converted to clauses and stored in the memory of the computer. The memory is a list structure indexed by the predicate letters, function symbols, and constant symbols occurring in each clause. A statement can be a very specific fact such as

STATEMENT: $COLOR(book, red)$

corresponding to the common attribute-object-value triple. A statement can also be a more general description of relations, such as:

STATEMENT: $(\forall x)(\forall A)(\forall B)[A \subseteq B \wedge x \in A \Rightarrow x \in B]$

meaning that if A is a subset of B and if x is an element of A, then x is an element of B.

Questions are also presented as statements of first-order logic. QUESTION is typed before the question. This question becomes a conjecture and QA3

attempts to prove the conjecture in order to answer *YES*. If the conjecture is not proved, QA3 attempts to prove the negation of this question in order to answer *NO*. The theorem-prover attempts a proof by refutation. During the process of searching for a proof, clauses that may be relevant to a proof are extracted from memory and utilized as axioms. If the question is neither proved nor disproved, then a *NO PROOF FOUND* answer is returned. ANSWER indicates an answer.

We now present a very simple dialogue with QA3. The dialogue illustrates a 'yes' answer, a 'no' answer, and an 'or' answer. Questions 4, 7, and 8 below illustrate questions where the answer is a term generated by the proof procedure. These kinds of answers will be called 'constructive' answers.

1. The first fact is 'Smith is a man.'
   STATEMENT: $MAN(Smith)$
   OK

The OK response from QA3 indicates that the statement is accepted, converted to a clause, and stored in memory.

2. We ask the first question, 'Is Smith a man?'
   QUESTION: $MAN(Smith)$
   ANSWER: *YES*

3. We now state that 'Man is an animal,' or, more precisely, 'If x is a man then x is an animal.'
   STATEMENT: $(\forall x)[MAN(x) \Rightarrow ANIMAL(x)]$
   OK

4. We now ask 'Who is an animal?' This question can be restated as 'Find some y that is an animal' or 'Does there exist a y such that y is an animal? If so, exhibit such a y.'
   QUESTION: $(\exists y)ANIMAL(y)$
   ANSWER: *YES*, $y = Smith$

The *YES* answer indicates that the conjecture $(\exists y)ANIMAL(y)$ has been proved (from statements 1 and 3 above). '$y = Smith$' indicates that '*Smith*' is an instance of y satisfying $ANIMAL(y)$ – i.e., $ANIMAL(Smith)$ is a theorem.

5. Fact: A robot is a machine.
   STATEMENT: $(\forall x)[ROBOT(x) \Rightarrow MACHINE(x)]$
   OK

6. Fact: Rob is a robot.
   STATEMENT: $ROBOT(Rob)$
   OK

7. Fact: No machine is an animal.

   STATEMENT: $(\forall x)[MACHINE(x) \Rightarrow \sim ANIMAL(x)]$

   OK

8. The question 'Is everything an animal?' is answered *NO*.
   A counterexample is exhibited – namely, Rob the robot.

   QUESTION: $(\forall x)ANIMAL(x)$

   ANSWER: $NO, x=Rob$

The answer indicates that $\sim ANIMAL(Rob)$ is a theorem. Note that a *NO* answer produces a counterexample for the universally quantified variable x. This is the dual of the construction of a satisfying instance for an existentially quantified variable in a question answered *YES*.

9. Fact: Either Smith is at work or Jones is at work.

   STATEMENT: $AT(Smith,work) \vee AT(Jones,work)$

   OK

10. Question: 'Is any one at work?'

    QUESTION: $(\exists x)(AT(x,work))$

    ANSWER: $YES, x=Smith$

    or $x=Jones$

From the previous statement it is possible to prove that someone is at work, although it is not possible to specify a unique individual.

Statements, questions, and answers can be more complex so that their corresponding English form is not so simple. Statements and questions can have many quantifiers and can contain functions. The answer can also contain functions. Consider the question 'Is it true that for all x there exists a y such that $P(x,y)$ is true?', where P is some predicate letter. Suppose QA3 is given the statement,

11. STATEMENT: $(\forall z)P(z,f(z))$

where f is some function. We ask the question

12. QUESTION: $(\forall x)(\exists y)P(x,y)$

    ANSWER: $YES, y=f(x)$

Notice that the instance of y found to answer the question is a function of x, indicating the dependence of y on x. Suppose that instead of statement 11 above, QA3 has other statements about P. An answer to question 12 might be

ANSWER: $NO, x=a$

where a is some instance of x that is a counterexample.

The term(s) that is the answer can be either a constant, a function, a variable, or some combination thereof. If the answer is a constant or a known function, then the meaning of the answer is clear. However, the answer may

be a Skolem function generated by dropping existential quantifiers. In this case, the answer is an object asserted to exist by the existential quantifier that generated the Skolem function. To know the meaning of this Skolem function, the system must exhibit the original input statement that caused the production of the Skolem function. Free variables in clauses correspond to universally quantified variables, so if the answer is a free variable, then any term satisfies the formula and thus answers the question.

Two more types of answers are *NO PROOF FOUND* and *INSUFFICIENT INFORMATION.* Suppose the theorem-prover fails to prove some conjecture and also fails to disprove the conjecture. If the theorem-prover runs out of time or space during either the attempted 'yes' proof or the attempted 'no' proof, then there is the possibility that some proof is possible if more time or space is available. The answer in this case is *NO PROOF FOUND.*

Now suppose both proof attempts fail without exceeding any time or space limitations. The theorem-proving strategy is complete so that if no time or space limitation halts the search for a proof and the conjecture is a logical consequence of the axioms, then a proof will be found. So we know that neither a 'yes' nor a 'no' answer is possible from the given statements. The answer returned is *INSUFFICIENT INFORMATION.* For example, suppose QA3 has no statements containing the predicate letter 'R':

QUESTION: $(\exists x)R(x)$

The negated question is the clause $\{\sim R(x)\}$, and no other clauses in the memory of QA3 can resolve with it. Thus the system will respond

ANSWER: *INSUFFICIENT INFORMATION.*

2. Constructing answers

The Resolution method of proving theorems allows us to produce correct constructive answers. This means that if, for example, $(\exists x)P(x)$ is a theorem then the proof procedure can find terms $t_1, t_2, \ldots, t_n$ such that $P(t_1) \vee P(t_2) \vee \ldots \vee P(t_n)$ is a theorem.

First, we shall present some examples of answer construction. After these examples we shall show how a proof by resolution can be used to generate an answer.

Examples of answer construction will be explained by means of the *ANSWER* predicate used by QA3 to keep track of instantiations. Consider the question

QUESTION: $(\exists y)ANIMAL(y)$

which is negated to produce the clause

$\{\sim ANIMAL(y)\}$.

The special literal, $ANSWER(y)$, is added to this clause to give

$\{\sim ANIMAL(y) \vee ANSWER(y)\}$.

The proof process begins with this clause. When the literal $ANIMAL(x)$ is resolved against the literal $\sim ANIMAL(y)$, the term y is instantiated to yield the term x. In the new clause resulting from this resolution, the argument of $ANSWER$ is then x. In the next resolution the argument of $ANSWER$ becomes Smith. We list the complete modified proof that terminates with the clause

$\{ANSWER(Smith)\}$.

1.	$\{\sim ANIMAL(y) \vee ANSWER(y)\}$	Modified negation of the question.
2.	$\{\sim MAN(x) \vee ANIMAL(x)\}$	Axiom fetched from memory.
3.	$\{\sim MAN(x) \vee ANSWER(x)\}$	From resolving 1 and 2.
4.	$\{MAN(Smith)\}$	Axiom fetched from memory.
5.	$\{ANSWER(Smith)\}$	'Contradiction' from 3 and 4 for $y=Smith$.

The argument of the $ANSWER$ predicate is the instance of y – namely, *Smith* – that answers the question. QA3 returns

ANSWER: YES, $y=Smith$.

This answer means, as will be explained later, that

$ANIMAL(Smith)$

is a theorem.

The $ANSWER$ literal is added to each clause in the negation of the question. The arguments of $ANSWER$ are the existentially quantified variables in the question. When a new clause is created, each $ANSWER$ literal in the new clause is instantiated in the same manner as any other literal from the parent clause. However, the $ANSWER$ literal is treated specially; it is considered to be invisible to resolution in the sense that no literal is resolved against it and it does not contribute to the length (size) of the clause containing it. We call a clause containing only $ANSWER$ literals an 'answer clause.' The search for an answer (proof) successfully terminates when an answer clause is generated. The addition of the $ANSWER$ predicate to the clauses representing the negation of the theorem does not affect the completeness of this modified proof procedure. The theorem-prover generates the same clauses, except for the $ANSWER$ predicate, as the conventional theorem-prover. Thus in this system an answer clause is equivalent to the empty clause that establishes a contradiction in a conventional system.

An answer clause specifies the sets of values that the existentially quantified variables in the question may take in order to preserve the provability of the question. The precise meaning of the answer will be specified in terms of a question Q that is proved from a set of axioms $B=\{B_1,B_2, \ldots, B_b\}$.

As an example illustrating some difficulties with Skolem functions, let the axioms B consist of a single statement,

STATEMENT: $(\forall z)(\exists w)P(z,w)$

Suppose this is converted to the clause

$$\{P(z,f(z))\},$$

where $f(z)$ is the Skolem function due to the elimination of the quantifier $(\exists w)$. We ask the question Q,

QUESTION: $(\forall y)(\exists x)P(y,x)$.

The negation of the question is $\sim Q$,

$$(\exists y)(\forall x)\sim P(y,x).$$

The clause representing $\sim Q$ is

$$\{\sim P(b,x)\},$$

where b is the constant (function of no variables) introduced by the elimination of $(\exists y)$. The proof, obtained by resolving these two clauses, yields the answer clause

$$\{ANSWER(f(b))\}.$$

The Skolem Function b is replaced by y, and the answer printed out is

ANSWER: $YES,\ x=f(y)$. (1)

At present in QA3 the Skolem function $f(y)$ is left in the answer. To help see the meaning of some Skolem function in the answer, the user can ask the system to display the original statement that, when converted to clauses, caused the generation of the Skolem function.

As an illustration, consider the following interpretation of the statement and question of this example. Let $P(u,v)$ be true if u is a person at work and v is this person's desk. Then the statement $(\forall z)(\exists w)P(z,w)$ asserts that every person at work has a desk, but the statement does not name the desk. The Skolem function $f(z)$ is created internally by the program during the process of converting the statement $(\forall z)(\exists w)P(z,w)$ into the clause $\{P(z,f(z))\}$. The function $f(z)$ may be thought of as the program's internal name for z's desk. (The term $f(z)$ could perhaps be written more meaningfully in terms of the descriptive operator i as '$iw.P(z,w)$,' i.e., 'the w such that $P(z,w)$', although w is not necessarily unique.)

The question $(\forall y)(\exists x)P(y,x)$ asks if for every person y there exists a corresponding desk. The denial of the question, $(\exists y)(\forall x)\sim P(y,x)$, postulates that there exists a person such that for all x, it is not the case that x is his desk. The Skolem function of no arguments, b, is also created internally by the

program as it generates the clause $\{\sim P(b,x)\}$. The function b is thus the program's internal name for the hypothetical person who has no desk.

The one-step proof merely finds that b does have a desk, namely $f(b)$. The user of the system does not normally see the internal clause representations unless he specifically requests such information. If the term $f(b)$ that appears in the answer clause were given to the user as the answer, e.g. $YES, x=f(b)$, the symbols f and b would be meaningless to him. But the program remembers that b corresponds to y, so b is replaced by y, yielding a slightly more meaningful answer, $YES, x=f(y)$. The user then knows that y is the same y he used in the question. The significance of the Skolem function f is slightly more difficult to express. The program must tell the user where f came from. This is done by returning the original statement $(\forall z)P(z,f(z))$ to the user (alternatively, the descriptive operator could be used to specify that $f(z)$ is $iw.P(z,w)$). As a rule, the user remembers, or has before his eyes, the question, but the specific form of the statements (axioms) is forgotten. In this very simple example the meaning of f is specified completely in terms of the question predicate P, but in general the meanings of Skolem functions will be expressed in terms of other predicates, constants, etc.

We will now show how to construct an 'answer statement', and then we will prove that the answer statement is a logical consequence of the axiom clauses. The user may require that an answer statement be exhibited, in order better to understand a complicated answer.

Consider a proof of question Q from the set of axioms $B=\{B_1,B_2,\ldots,B_b\}$. B logically implies Q if and only if $B\wedge\sim Q$ is unsatisfiable. The statement $B\wedge\sim Q$ can be written in prenex form $PM(Y,X)$, where P is the quantifier prefix, $M(Y,X)$ is the matrix, $Y=\{y_1,y_2,\ldots,y_u\}$ is the set of existentially quantified variables in P, and $X=\{x_1,x_2,\ldots,x_e\}$ is the set of universally quantified variables in P.

Eliminating the quantifier prefix P by introducing Skolem functions to replace existential quantifiers and dropping the universal quantifiers produces the formula $M(U,X)$. Here U is the set of terms $\{u_1,u_2,\ldots,u_u\}$, such that for each existentially quantified variable y_i in P, u_i is the corresponding Skolem function applied to all the universally quantified variables in P preceding y_i. Let $M(U,X)$ be called S. The statement $B\wedge\sim Q$ is unsatisfiable if and only if the corresponding statement S is unsatisfiable. Associated with S is a Herbrand Universe of terms H that includes X, the set of free variables of S. If $\phi=\{t_1/x_1,t_2/x_2,\ldots,t_n/x_n\}$ represents a substitution of terms $t_1, t_2, \ldots, t_n$ from H for the variables $x_1, x_2, \ldots, x_n$, then $S\phi$ denotes the instance of S over H formed by substituting the terms $t_1, t_2, \ldots, t_n$ from H for the corresponding variables $x_1, x_2, \ldots, x_n$ in S.

Let S_i represent a variant of S, i.e., a copy of S with the free variables renamed. Let the free variables be renamed in such a way that no two variants S_i and S_j have variables in common. By the Skolem-Löwenheim-Gödel theorem (Robinson 1967), S is unsatisfiable if and only if there exists an

instance of a finite conjunction of variants of S that is truth-functionally unsatisfiable. A resolution theorem prover can be interpreted as proving S unsatisfiable by finding such a finite conjunction.

Suppose the proof of Q from B finds the conjunction $S_1 \wedge S_2 \wedge \ldots \wedge S_k$ and the substitution θ such that

$$(S_1 \wedge S_2 \wedge \ldots \wedge S_k)\theta$$

is truth-functionally unsatisfiable. Let F_0 denote the formula $(S_1 \wedge S_2 \wedge \ldots \wedge S_k)\theta$. Let L be the conjunction of variants of $M(Y,X)$,

$$L = M(Y_1,X_1) \wedge M(Y_2,X_2) \wedge \ldots \wedge M(Y_k,X_k)$$

and let λ be the substitution of Skolem function terms for variables such that

$$\begin{aligned} L\lambda &= M(U_1,X_1) \wedge M(U_2,X_2) \wedge \ldots \wedge M(U_k,X_k) \\ &= S_1 \wedge S_2 \wedge \ldots \wedge S_k. \end{aligned}$$

Thus $L\lambda\theta = F_0$.

Before constructing the answer statement, observe that the Skolem functions of F_0 can be removed as follows. Consider the set $U = \{u_1,u_2, \ldots, u_u\}$ of Skolem-function terms in S. Find in F_0 one instance, say u_1', of a term in U. Select a symbol, z_1, that does not occur in F_0. Replace every occurrence of u_1' in F_0 by z_1, producing statement F_1. Now again apply this procedure to F_1, substituting a new variable throughout F_1 for each occurrence of some remaining instance of a Skolem-function term in F_1, yielding F_2. This process can be continued until no further instances of terms from U are left in F_n, for some n.

The statement F_i for $0 \leqslant i \leqslant n$ is also truth-functionally unsatisfiable for the following reasons. Consider any two occurrences of atomic formulae, say m_a and m_b, in F_0. If m_a and m_b in F_0 are identical, then the corresponding two transformed atomic formulae m_{a1} and m_{b1} in F_2 are identical. If m_a and m_b are not identical, then m_{a1} and m_{b1} are not identical. Thus, F_1 must have the same truth table, hence truth value, as F_0. This property holds at each step in the construction, so $F_0, F_1, \ldots, F_n$ must each be truth-functionally unsatisfiable.

This term replacement operation can be carried out directly on the substitutions, i.e., for each statement F_i, $0 \leqslant i \leqslant n$, there exists a substitution σ_i such that $F_i = L\sigma_i$. We prove this by showing how such a σ_i is constructed. Let $\sigma_0 = \lambda\theta = \{t_1/v_1, t_2/v_2, \ldots, t_p/v_p\}$.

By definition, $F_0 = L\sigma_0$. Let t_j' denote the term formed by replacing every occurrence of u_1' in t_j by z_1. The substitution $\sigma_1 = \{t_1'/v_1, t_2'/v_2, \ldots, t_p'/v_p\}$ applied to L yields F_1, i.e., $F_1 = L\sigma_1$. Similarly one constructs σ_i and shows, by induction, $F_i = L\sigma_i$, for $0 \leqslant i \leqslant n$.

Now let us examine some of the internal structure of F_0. Assume that $S = M(U,X)$ is formed as follows. The axioms may be represented as $P_B B(Y_B,X_B)$, where P_B is the quantifier prefix, Y_B is the set of universally-

quantified variables, and X_B is the set of existentially-quantified variables. These axioms are converted to a set of clauses denoted by $B(Y_B,U_B)$, where U_B is the set of Skolem-function terms created by eliminating X_B.

The question may be represented as $P_Q Q(Y_Q,X_Q)$, where P_Q is the quantifier prefix, Y_Q is the set of universally-quantified variables, and X_Q is the set of existentially-quantified variables. Assume that the variables of the question are distinct from the variables of the axioms. The negation of the question is converted into a set of clauses denoted by $\sim Q(U_Q,X_Q)$, where U_Q is the set of Skolem-function terms created by eliminating Y_Q. The function symbols in U_Q are distinct from the function symbols in U_B. Thus $M(U,X)=[B(Y_B,U_B) \wedge \sim Q(U_Q,X_Q)]$. Now let $L_B=[B(Y_{B1},X_{B1}) \wedge B(Y_{B2},X_{B2}) \wedge \ldots \wedge B(Y_{Bk},X_{Bk})]$ and let $\sim L_Q=[\sim Q(Y_{Q1},X_{Q1}) \wedge \sim Q(Y_{Q2},X_{Q2}) \wedge \ldots \wedge \sim Q(Y_{Qk},X_{Qk})]$. Thus $L=L_B \wedge \sim L_Q$.

Observe that one can construct a sequence of statements $F_0, F_1', \ldots, F_m'$ similar to $F_0, F_1, \ldots, F_n$ in which the only terms replaced by variables are instances of terms in U_Q. This construction terminates when for some m the set of clauses F_m' contains no further instances of terms in U_Q. By the same argument given earlier for the formulas F_i, each formula F_i' is truth-functionally unsatisfiable. Similarly one can construct a sequence of substitutions $\sigma_0, \sigma_1', \ldots, \sigma_m'$ such that $L\sigma_i'=F_i'$ for $0 \leqslant i \leqslant m$. Let $\sigma=\sigma_m'$. Substitute σ into L_Q, forming

$$L_Q\sigma=[Q(Y_{Q1},X_{Q1})\sigma \vee Q(Y_{Q2},X_{Q2})\sigma \vee \ldots \vee Q(Y_{Qk},X_{Qk})\sigma].$$

Since σ replaces the elements of Y_{Qj} by variables, let the set of variables Z_{Qj} denote $Y_{Qj}\sigma$. Thus

$$L_Q\sigma=[Q(Z_{Q1},X_{Q1}\sigma) \vee Q(Z_{Q2},X_Q\sigma) \vee \ldots \vee Q(Z_{Qk},X_{Qk}\sigma)].$$

Now, let Z be the set of all variables occurring in $L_Q\sigma$. The *answer statement* is defined to be $(\forall Z)\, L_Q\sigma$. In its expanded form the answer statement is

$$(\forall Z)[Q(Z_{Q1},X_{Q1}\sigma) \vee Q(Z_{Q2},X_{Q2}\sigma) \vee \ldots \vee Q(Z_{Qk},X_{Qk}\sigma)]. \quad (2)$$

We now prove that the answer statement is a logical consequence of the axioms in their clausal form. Suppose not, then $B(Y_B,U_B) \wedge \sim(\forall Z)L_Q\sigma$ is satisfiable, thus $B(U_B,X_B) \wedge (\exists Z) \sim L_Q\sigma$ is satisfiable, implying that the conjunction of its instances $L_B\lambda \wedge (\exists Z) \sim L_Q\sigma$ is satisfiable. Now drop the existential quantifiers $(\exists Z)$. Letting the elements of Z in $\sim L_Q\sigma$ denote a set of constant symbols or Skolem functions of no arguments, the resulting formula $L_B\lambda \wedge \sim L_Q\sigma$ is also satisfiable.

Note that $L_B\sigma$ is an instance of $L_B\lambda$. To see this, let λ_B be the restriction of λ to variables in L_B. Thus, $L_B\lambda=L_B\lambda_B$. Suppose $\theta=\{r_1/w_1, r_2/w_2, \ldots, r_n/w_n\}$. Recall that σ is formed from $\lambda\theta$ by replacing in the terms of $\lambda\theta$ occurrences of instances u_q' of 'question' Skolem terms by appropriate variables. (The 'axiom' Skolem functions are distinct from question Skolem functions and occur only in the terms of λ_B.) Thus no such u_q' is an instance of an axiom Skolem term, therefore each occurrence of each such u_q' in $\lambda_B\theta$ must arise

from an occurrence of u_q' in some r_j in θ. It follows then that $L_B\sigma = L_B\lambda_B\phi$ where $\phi = \{r_1'/w_1, r_2'/w_2, \ldots, r_n'/w_n\}$ is formed from θ by replacing each u_q' in each r_j by an appropriate variable. Since $L_B\lambda = L_B\lambda_B$, $L_B\lambda\phi = L_B\sigma$. Since the only free variables of $L_B\lambda \wedge \sim L_Q\sigma$ occur in $L_B\lambda$, $[L_B\lambda \wedge \sim L_Q\sigma]\ \phi = L_B\lambda\phi \wedge \sim L_Q\sigma$.

The formula $L_B\lambda \wedge \sim L_Q\sigma$ logically implies all of its instances, in particular the instance $L_B\lambda\phi \wedge \sim L_Q\sigma$. Thus, if $L_B\lambda \wedge \sim L_Q\sigma$ is satisfiable, its instance $L_B\lambda\phi \wedge \sim L_Q\sigma$ is satisfiable. Since $[L_B\lambda\phi \wedge \sim L_Q\sigma] = [L_B\sigma \wedge \sim L_Q\sigma] = [L_B \wedge \sim L_Q]\sigma = L\sigma = F_m'$ for some m, F_m' must be satisfiable. This contradicts our earlier result that F_m' is truth-functionally unsatisfiable, and thus proves that the answer statement is a logical consequence of the axioms.

We make one further refinement of the answer statement (2). It is unnecessary to include the jth disjunct if $X_{Qj}\sigma = X_{Qj}$, i.e., if σ does not instantiate X_{Qj}. Without loss of generality, we can assume that for $r \leqslant k$, the last $k-r$ disjuncts are not instantiated, i.e.,

$$X_{Qr+1}\sigma = X_{Qr+1},\ X_{Qr+2}\sigma = X_{Qr+2}, \ldots, X_{Qk}\sigma = X_{Qk}.$$

Then the stronger answer statement

$$(\forall Z)[Q(Z_{Q1}, X_{Q1}\sigma) \vee Q(Z_{Q1}, X_{Q2}\sigma) \vee \ldots \vee Q(Z_{Qr}, X_{Qr}\sigma)] \qquad (3)$$

is logically equivalent to (2). (Since the matrix of (3) is a sub-disjunct of (2), (3) implies (2). If $j \leqslant r$, the jth disjunct of (2) implies the jth disjunct of (3). If $r < j \leqslant k$, the jth disjunct of (2) implies all of its instances, in particular all disjuncts of (3).)

The *ANSWER* predicate provides a simple means of finding the instances of Q in (3). Before the proof attempt begins, the literal $ANSWER(X_Q)$ is added to each clause in $\sim Q(U_Q, X_Q)$. The normal resolution proof procedure then has the effect of creating new variants of X_Q as needed. The jth variant, $ANSWER(X_{Qj})$, thus receives the instantiations of $\sim Q(U_{Qj}, X_{Qj})$. When a proof is found, the answer clause will be

$$\{ANSWER(X_{Q1}\theta) \vee ANSWER(X_{Q2}\theta) \ldots \vee ANSWER(X_{Qr}\theta)\}.$$

Variables are then substituted for the appropriate Skolem functions to yield

$$\{ANSWER(X_{Q1}\sigma) \vee ANSWER(X_{Q2}\sigma) \ldots \vee ANSWER(X_{Qr}\sigma)\}.$$

Let $X_{Qj} = \{x_{j1}, x_{j2}, \ldots, x_{jm}\}$.

Let σ restricted to X_{Qj} be $\{t_{j1}/x_{j1}, t_{j2}/x_{j2}, \ldots, t_{jm}/x_{jm}\}$.

The answer terms printed out by QA3 are

$$[x_{11} = t_{11} \text{ and } x_{12} = t_{12} \ldots \text{ and } x_{1m} = t_{1m}] \qquad (4)$$

or $[x_{21} = t_{21}$ and $x_{22} = t_{22} \ldots$ and $x_{2m} = t_{2m}]$

$\vdots$

or $[x_{r1} = t_{r1}$ and $x_{r2} = t_{r2} \ldots$ and $x_{rm} = t_{rm}]$.

According to (3), all the free variables in the set Z that appear in the answer

are universally quantified. Thus any two occurrences of some free variable in two terms must take on the same value in any interpretation of the answer.

In the example given above, whose answer (1) had the single answer term $f(y)$, the complete answer statement is

$$(\forall y)P(y,f(y)).$$

In section 3.3 we present two more examples. The answer in the second example has four answer terms, illustrating the subcase of (4),

$$[x_{11}=t_{11} \text{ and } x_{12}=t_{12}]$$
or $$[x_{21}=t_{21} \text{ and } x_{22}=t_{22}].$$

The answer statement proved can sometimes be simplified. For example, consider

QUESTION: $(\exists x)P(x)$
ANSWER: $YES, x=a$
or $x=b,$

meaning that the answer statement proved is

$$[P(a) \vee P(b)].$$

Suppose it is possible to prove $\sim P(b)$ from other axioms. Then a simpler answer is provable, namely

ANSWER: $YES, x=a.$

3. Processes described as a state transformation

In some of the applications of QA3 mentioned in section 5 it is necessary to solve problems of the kind: 'Find a sequence of actions that will achieve some goal.' One method for solving this type of problem is to use the notion of transformations of states. We show here how processes involving changes of state can be described in first-order logic and how this formalism is used. The process of finding the values of existentially quantified variables by theorem-proving can be used to find the sequence of actions necessary to reach a goal.

The basic mechanism is very simple. A first-order logic function corresponds to an action or operator. This function maps states into new states. An axiom takes the following form:

$$P(s_1) \wedge (f(s_1)=s_2) \Rightarrow Q(s_2)$$

where

s_1 is the initial state
$P(s_1)$ is a predicate describing the initial state
$f(s_1)$ is a function (corresponding to an action)
s_2 is the value of the function, the new state
$Q(s_2)$ is a predicate describing the new state.

The equality can be eliminated, giving

$$P(s_1) \Rightarrow Q(f(s_1)).$$

As an example, consider how one might describe the movements of a robot Each state will correspond to one possible position of the robot. Consider the statement 'If the robot is at point a in some state s_1, and performs the action of moving from a to b, then the robot will be at position b in some resulting state s_2.' The axiom is

$$(\forall s_1)(\forall s_2)[AT(a,s_1) \wedge (move(a,b,s_1) = s_2) \Rightarrow AT(b,s_2)].$$

The function *move* (a,b,s_1) is the action corresponding to moving from a to b. The predicate $AT(a,s_1)$ is true if and only if the robot is at point a in state s_1. The predicate $AT(b,s_2)$ is true if and only if the robot is at point b in state s_2.

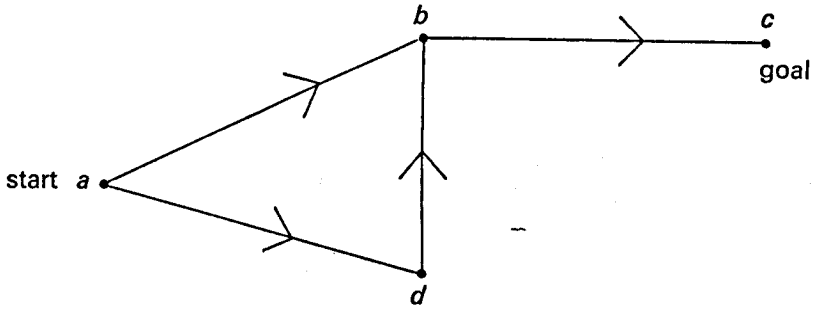

Figure 1

Now consider an example showing how the theorem-prover can be used to find a sequence of actions that reach a goal. The robot starts at position a in initial state s_0. From a he can move either to b or d. From b he can move to c. From d he can move to b. The allowed moves are shown in figure 1.

The axioms are:

A_1. $AT(a,s_0)$
A_2. $(\forall s_1)[AT(a,s_1) \Rightarrow AT(b,move(a,b,s_1))]$
A_3. $(\forall s_2)[AT(a,s_2) \Rightarrow AT(d,move(a,d,s_2))]$
A_4. $(\forall s_3)[AT(b,s_3) \Rightarrow AT(c,move(b,c,s_3))]$
A_5. $(\forall s_4)[AT(d,s_4) \Rightarrow AT(b,move(d,b,s_4))]$

Axiom A_1 states that the robot starts at position a in State s_0. Axioms A_2, A_3 A_1, and A_1 describe the allowed moves.

We now ask for a sequence of actions that will move the robot to position c. We present this question in the form 'Does there exist a state in which the robot is at position c?'

QUESTION: $(\exists s)AT(c,s)$
ANSWER: YES, $s = move(b,c,move(a,b,s_0))$

By executing this resulting function $move(b,c,move(a,b,s_0))$ our hypothetical robot could effect the desired sequence of actions. The normal order of

evaluating functions, starting with the innermost and working outward, gives the order of performing the actions: move from a to b and then move from b to c. In general, this technique of function composition can be used to specify sequences of actions.

The proof of the answer by resolution is given below, with comments. The negation of the question is $(\forall s) \sim AT(c,s)$, and the refutation process finds, by instantiation, the value of s that leads to a contradiction. The successive instantiations of s appear as arguments of the special predicate, $ANSWER$. The constants are a, b, c, and s_0. The free variables are s, s_1, s_2, s_3, and s_4.

Proof

1. $\{\sim AT(c,s \vee ANSWER(s))\}$	Negation of question
2. $\{\sim AT(b,s_3) \vee AT(c,move(b,c,s_3))\}$	Axiom A_4
3. $\{\sim AT(b,s_3) \vee ANSWER(move(b,c,s_3))\}$	From resolving 1 and 2
4. $\{\sim AT(a,s_1) \vee AT(b,move(a,b,s_1))\}$	Axiom A_2
5. $\{\sim AT(a,s_1) \vee ANSWER(move(b,c,move(a,b,s_1)))\}$	From resolving 3 and 4
6. $\{AT(a,s_0)\}$	Axiom A_1
7. $\{ANSWER(move(b,c,move(a,b,s_0)))\}$	From resolving 5 and 6

Note that the process of proving the theorem corresponds to starting at the goal node c and finding a path back to the initial node a.

Consider a second example. Two players p_1 and p_2 play a game. In some state s_1, player p_1 is either at position a or position b.

$B1.$ $AT(p_1,a,s_1) \vee AT(P_1,b,s_1)$.

If in state s_1, player p_2 can move anywhere.

$B2.$ $(\forall y)AT(p_2,y,move(p_2,y,s_1))$

The position of player p_1 is not affected by p_2's movement.

$B3.$ $(\forall x)(\forall y)(\forall s)[AT(p_1,x,s) \Rightarrow AT(p_1,x,move(p_2,y,s))]$

Does there exist some state (sequence) such that p_1 and p_2 are together?

QUESTION: $(\exists x)(\exists s)[AT(p_1,x,s) \vee AT(p_2,x,s)]$

ANSWER: YES, $[x=a$ and $s=move(p_2,a,s_1)]$

or

$[x=b$ and $s=move(p_2,b,s_1)]$

This answer indicates that two meeting possibilities exist; either (1) player p_1 is at position a and player p_2 moves to a, meeting p_1 at a, or (2) player p_1 is at position b and player p_2 moves to b, meeting p_1 at b. However, the 'or' answer indicates that we do not know which one move will lead to a meeting.

The 'or' answer is due to the fact that Axiom B_1 did not specify player p_1's position. The answer statement that has been proved is

$$[AT(p_1,a,move(p_2,a,s_1)) \wedge AT(p_2,a,move(p_2,a,s_1))] \vee [AT(p_1,b,move(p_2,b,s_1)) \wedge AT(p_2,b,move(p_2,b,s_1))].$$

Proof

1.	$\{\sim AT(p_1,x,s) \vee \sim AT(p_2,x,s) \vee ANSWER(x,s)\}$	Negation of question
2.	$\{AT(p_2,y,move(p_2,y,s))\}$	Axiom $B2$
3.	$\{\sim AT(p_1,x,move(p_2,x,s_1)) \vee ANSWER(x,move(p_2,x,s_1))\}$	From 1, 2
4.	$\{\sim AT(p_1,x,s) \vee AT(p_1,x,move(p_2,y,s))\}$	Axiom $B3$
5.	$\{\sim AT(p_1,y,s_1) \vee ANSWER(y,move(p_2,y,s_1))\}$	From 3, 4
6.	$\{AT(p_1,a,s_1) \vee AT(p_1,b,s_1)\}$	Axiom $B1$
7.	$\{AT(p_1,b,s_1) \vee ANSWER(a,move(p_2,a,s_1))\}$	From 5, 6
8.	$\{ANSWER(a,move(p_2,a,s_1)) \vee ANSWER(b,move(p_2,b,s_1))\}$	From 5, 7

It is possible to formalize other general problem-solving tasks in first-order logic, so that theorem-proving methods can be used to produce solutions. For a discussion of formalizations of several general concepts including cause, 'can', knowledge, time, and situations, see McCarthy and Hayes (1969).

4. PROGRAM ORGANIZATION

The organization of the question-answering program QA3 differs from that of a 'pure' theorem-proving program in some of the capabilities it emphasizes: a proof strategy intended for the quick answering of easy questions even with a large data base of axioms, a high level of interaction between the user and both the question-answering program and the data base in a suitable command language, and some flexibility in the question-answering process so that the program can be fitted to various applications. In this section we describe the principal features of the system.

1. Program control

The user can control the proof process in several ways.

1. The user can request a search for just a 'yes' answer, instead of both 'yes' and 'no'.
2. The user can request the program to keep trying, by increasing its effort if no proof is found within preset limits. This lets QA3 search for a more difficult proof.
3. When a proof is found it can be printed out. Included with the proof are statistics on the search: the number of clauses generated, the number of clauses subsumed out of the number attempted, the number of successful resolutions out of the number attempted, and the number of successful factors generated out of the number attempted.

4. The user can request that the course of the search be exhibited as it is in progress by printing out each new clause as it is generated or selected from memory, along with specified information about the clause.
5. The user can request that existentially quantified variables in the question be not traced.
6. The user can designate predicates and functions that are to be evaluated by LISP programs. For example, the predicate $1 \leqslant 2$ might be evaluated by LISP to yield the truth value *T*. This feature also allows the transfer of control to peripheral devices.
7. Parameters controlling the proof strategy, such as degree and set of support are accessible to the more knowledgeable user.
8. A number of editing facilities on the clauses in memory are useful:
   (a) A new axiom can be entered into memory,
   (b) An axiom in memory can be deleted, and
   (c) The axioms containing any predicate letter can be listed.

2. Special uses of the theorem-prover

'The theorem-prover' refers to a collection of LISP functions used during the theorem-proving process – e.g. RESOLVE, FACTOR, PROVE, PRENEX, CHECKSUBSUMPTION, etc.

The management of the data in memory is aided by the theorem-prover. A statement is stored in memory only if it is neither a tautology nor a contradiction. A new clause is not stored in memory if there already exists in memory another clause of equal length or shorter length that subsumes the new clause. Two other acceptance tests are possible although they are not now implemented. A statement given the system can be checked for consistency with the current data base by attempting to prove the negation of the statement. If the statement is proved inconsistent, it would not be stored. As another possible test, the theorem-prover could attempt to prove a new statement in only 1 or 2 steps. If the proof is sufficiently easy, the new statement could be considered redundant and could be rejected.

The theorem-prover can also be used to simplify the answer, as described in section 3.

3. Strategy

The theorem-proving strategy used in QA3 is similar to the unit-preference strategy, using an extended set-of-support and subsumption.

The principal modification for the purposes of the question-answering system is to have two sets of clauses during an attempted proof. The first set, called 'Memory', contains all the statements (axioms) given the system. The second set, called 'Clauselist' is the active set of clauses containing only

the axioms being used in the current proof attempt and the new clauses being generated. Clauselist is intended to contain only the clauses most relevant to the question.

There is a high cost, in computer time and space, for each clause actively associated with the theorem-prover. The cost is due to the search time spent when the clause is considered as a candidate for resolution, factoring, or subsumption, and the extra space necessary for book-keeping on the clause. Since most clauses in Memory are irrelevant to the current proof, it is undesirable to have them in Clauselist, unnecessarily consuming this time and space. So the basic strategy is to work only on the clauses in Clauselist, periodically transferring new, possibly relevant clauses from Memory into Clauselist. If a clause that cannot lead to a proof is brought into Clauselist, this clause can generate many unusable clauses. To help avoid this problem the strategy is reluctant to enter a non-unit clause into Clauselist.

The proof strategy of the program is modified frequently, but we shall present an approximate overview of the proof strategy. When a question is asked, Clauselist will initially contain only the negation of the question, which is the set-of-support. A modified unit preference strategy is followed on Clauselist, using a bound on degree. As this strategy is being carried out, clauses from Memory that resolve with clauses in Clauselist are added to Clauselist. This strategy is carried out on Clauselist until no more resolutions are possible for a given degree bound.

Finally, the bound is reached. Clauselist, with all of its book-keeping, is temporarily saved. If the theorem-prover was attempting a 'yes' answer, it now attempts a 'no' answer. If attempting a 'no' answer, it also saves the 'no' Clauselist, and returns a *NO PROOF FOUND* answer. The user may then continue the search requesting CONTINUE. If the bound is not reached in either the yes or no case, the *INSUFFICIENT INFORMATION* answer is returned. The strategy has the following refinements:

1. After a newly created unit fails to resolve with any units in Clauselist, it is checked against the units in Memory for a contradiction. This helps to find short proofs quickly.
2. Frequently, in the question-answering applications being studied, a proof consists of a chain of applications of two-clauses, i.e., clauses of length two. Semantically it usually means that set-membership of some element is being found by chaining through successive supersets or subsets. To speed up this process, a special fast section is included that resolves units in Clauselist with two-clauses in Memory. Our experience so far is that this heuristic is worthwhile.
3. Each new clause generated is checked to see if it is subsumed by another shorter clause in Clauselist. All longer clauses in Clauselist are checked to see if they are subsumed by the new clause. The longer subsumed clauses are deleted.

4. Hart's theorem (1965) shows how binary resolution can generate redundant equivalent proofs. Equivalent proofs are eliminated from the unit section. Wos terms this property, 'Singly-connectedness'. Currently this has not yet been implemented for the non-unit section.
5. An extended set-of-support is used, allowing pairs of clauses in Clauselist but not in the set-of-support to resolve with one another up to a level of 2.
6. The sets, Memory and Clauselist, are indexed to facilitate search. The clauses in Memory are indexed by predicate letters and, under each predicate letter, by length. The clauses in Clauselist are indexed by length.

   In searching Memory for relevant clauses to add to Clauselist, clauses already in Clauselist are not considered. The clauses of each length are kept on a list, with new clauses being added at the end of the list. Pointers, or place-keepers, are kept for these lists, and are used to prevent reconsidering resolving two clauses and also to prevent generating equivalent proofs.

   The strategy is 'complete' in the sense that it will eventually find any proof that exists within the degree and space bound.

5. PERFORMANCE OF QA3

1. Applications

The program has been tested on several question sets used by earlier question-answering programs. In addition, QA3 is now being used in other applications. The subjects for the first question set given QA2, reported in Green and Raphael (1968), consisted of some set-membership, set-inclusion, part-whole relationship and similar problems.

Raphael's SIR (1964b) program gave a similar but larger problem set also having the interesting feature of requiring facts or axioms from several subjects to interact in answering a question. SIR used a different subroutine to answer each type of question, and when a new relation was added to the system, not only was a new subroutine required to deal with that relation but also changes throughout the system were usually necessary to handle the interaction of the new relation with the previous relations. This programming difficulty was the basic obstacle in enlarging SIR. Raphael proposed a 'formalized question-answerer' as the solution. QA3 was tested on the SIR problem set with the following results: in two hours of sitting at the teletype all the facts programmed into or told to SIR were entered into the QA3 memory as axioms of first-order logic and QA3 answered essentially all the questions answered by SIR. The questions skipped used the special SIR heuristic, the 'exception principle'. It was possible to translate, as they were read, questions and facts stated in SIR's restricted English into first-order logic.

Slagle, in his paper on DEDUCOM, a question-answering system (1965), presented a broader, though less interactive, problem set consisting of gathered questions either answered by programs of, or else proposed by, Raphael (1964a), Black (1964), Safier (1963), McCarthy (1963), Cooper (1964), and Simon (1963). Included in this set were several examples of sequential processes, including one of McCarthy's End Game Questions (1963), Safier's Mikado Question (1963), McCarthy's Monkey-and-Bananas Question (1963), and one of Simon's State Description Compiler Questions (1963). Using the technique discussed in section 3.3 to describe processes, it was possible to axiomatize all the facts and answer all the questions printed in Slagle's paper. Furthermore, QA3 overcame some of the defects of DEDUCOM: QA3 could answer all answerable questions, the order of presenting the axioms did not affect its ability to answer questions, and no redundant facts were required. QA3 was then tested on the entire set of twenty-three questions presented in Cooper (1964). QA3 correctly answered all the questions, including four not answered by Cooper's program and sixteen not answered by DEDUCOM.

QA3 also solved the Wolf, Goat, and Cabbage puzzle in which a farmer must transport the wolf, goat, and cabbage across the river in a boat that can hold only himself and one other. The wolf cannot be left alone with the goat and the goat cannot be left alone with the cabbage.

In all of the problems mentioned above, QA3 was given the facts and questions in first-order logic. Raphael's program and Cooper's program used a restricted English input.

Using the English-to-logic translator developed by Coles (1968), Coles and Raphael have begun studying some medical question-answering applications of QA3.

QA3 is being tested in the Stanford Research Institute Automaton (robot) on problem-solving tasks.

2. Limitations

A few limitations should be emphasized. Firstly, QA3 is still not a finished system. One very important feature that is missing is the automatic handling of the equality relation, and this is not a trivial problem. Without an automatic equality capability, QA3 is very awkward on certain problems that are conveniently stated in terms of equality. The equality relation is but one instance of other 'higher-order' concepts (e.g. set theory) that either (i) cannot be described in first-order logic, or (ii) require some meta-level operations such as an axiom schema, or (iii) are awkward and impractical in first-order logic. However, it is not yet clear just what are the practical limitations of a first-order logic system having suitable 'tricks'.

One of the virtues of QA3 is that relatively subject-independent heuristics are used. All subject dependence comes from the particular axioms stored in memory, the theorem being proved, and the particular representation chosen

for each statement. This adds elegance and generality, yet yields a reasonably powerful system. However, for harder problems it may be necessary to be able to add subject-dependent search heuristics, or 'advice' for particular problems. Such an advice-taking capability will require a flexible and easily modifiable search strategy.

The particular heuristics used in QA3 are experimental and have not been thoroughly tested in question-answering applications (although the changes and heuristics added appear to have improved the system). As each modification of the strategy was added, the performance did improve on a particular class of problems. To help remedy some of this uncertainty several measures of performance are now automatically printed out after each question and will be used to evaluate questionable heuristics.

Another qualification is that the questions and subjects investigated were chosen from conjectured test problems or else from test problems used by other question-answering or problem-solving systems. This facilitates comparison, but does not necessarily indicate performance on more practical problems.

The new and more difficult applications being considered might lead to a better understanding of the exact limitations of QA3, or of theorem-proving techniques, for question-answering.

3. Performance

To answer any of the questions mentioned above, QA3 requires from a few seconds to a few minutes. We can roughly measure the problem-solving capacity of QA3 by giving the depth of search allowed and the free space available for storing clauses produced in searching for a proof. The space available for storing clauses produced during a proof typically allows a few hundred clauses to be stored. The depth of search is given by degree bound, normally set at 10. It is interesting to note that the many 'common sense' reasoning problems mentioned herein were within these bounds of QA3, and thus were not difficult proofs, compared to some of the mathematical proofs attempted by theorem-provers.

Acknowledgements

I would like to acknowledge the advice and guidance of Dr Bertram Raphael and Professor John McCarthy, as well as help from Mr. R. Kowalski in correcting an earlier draft of section 3; also the programming and ideas of Robert A. Yates.

The work reported here was supported under Contract AF 30(602)–4147, Rome Air Development Center, and the Advanced Research Projects Agency, Advanced Sensors Group; and also under Contract No. AF 19(628)–5919, Air Force Cambridge Research Laboratories.

REFERENCES

Black, F.S. (1964) A deductive question-answering system. Ph.D. thesis, Harvard.

Coles, L. S. (1968) An on-line question-answering system with natural language and pictorial input. *Proceedings 23rd ACM National Conference.*

Cooper, D. C. (1966) Theorem proving in computers, *Advances in programming and non-numerical computation* (ed. Fox, L.). London: Pergamon Press.

Cooper, W.S. (1964) Fact retrieval and deductive question answering information retrieval systems, *J. Ass. comput. Mach.*, **11**, 117–37.

Davis, M. (1963) Eliminating the irrelevant from mechanical proofs. *Annual symposia in applied mathematics XIX.* Providence, Rhode Island: American Mathematical Society.

Green, B.F., Jr., Wolf, A.K., Chomsky, C. & Laughery, K. (1963) BASEBALL: *an automatic question answerer. Computers and thought* (eds Feigenbaum, E.A., & Feldman, J.). New York: McGraw-Hill.

Green, C.C., & Raphael, B. (1968) The use of theorem-proving techniques in question-answering systems. *Proceedings 23rd ACM National Conference.*

Hart, T.P. (1965) A useful algebraic property of Robinson's Unification Algorithm. Memo No. 91, AI Project, Project MAC, MIT.

Lindsay, R.K. (1963) Inferential memory as the basis of machines which understand natural language. *Computers and thought* (eds Feigenbaum, E.A. & Feldman, J.). New York: McGraw-Hill.

McCarthy, J. (1958) Programs with common sense. *Symposium mechanization of thought processes.* Teddington: Nat. Physical Lab.

McCarthy, J. (1963) Situations, actions & causal laws. *Stanford artificial intelligence project memorandum*, No. 2.

McCarthy, J. & Hayes, P. (1969) Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence 4*, p. 463–502 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Raphael, B. (1964a) A computer program which 'understands'. *Proc. FJCC* Washington, D.C.: Spartan Books.

Raphael, B. (1964b) SIR: a computer program for semantic information retrieval. MAC–TR2, Project MAC, MIT.

Robinson, J.A. (1965) A machine-oriented logic based on the resolution principle. *J. Ass. comput. Mach.*, **12**, 23–41.

Robinson, J.A. (1967) A review of automatic theorem-proving. *Annual symposia in applied mathematics XIX.* Providence, Rhode Island: American Mathematical Society.

Safier, F. (1963) The Mikado as an advice taker problem. *Stanford artificial intelligence project memorandum*, no. 3.

Safier, F. (1965) Simple Simon. *Stanfordartificial intelligence project memorandum*, no. 35.

Simon, H. (1963) Experiments with a heuristic compiler, *J. Ass. comput. Mach.*, **10**, 493–50

Slagle, J.R. (1965) Experiments with a deductive, question-answering program. *Communications of the ACM* **8**, 792–8.

Wos, L., Carson, D. F. & Robinson, G.A. (1964) The unit preference strategy in theorem proving. *AFIPS* **26**, 615–21, Fall, J.C.C. Washington, D.C.: Spartan Books.

Wos, L. T., Carson, D.F. & Robinson, G.A. (1965) Efficiency and completeness of the set-of-support strategy in theorem-proving. *J. Ass. comput. Mach.*, **12**, 536–41.

Wos, L.T., Robinson, G.A., Carson, D.F. & Shalla, L. (1967) The concept of demodulation in theorem-proving. *J. Ass. comput. Mach.*, **14**, 698–709.