THE Q* ALGORITHM - A SEARCH STRATEGY FOR A DEDUCTIVE QUESTION-ANSWERING SYSTEM

Jack Minker
Daniel H. Fishman
James R. McSkimin

The University of Maryland
Computer Science Center
College Park, Maryland 20742

Abstract

An approach for bringing semantic, as well as syntactic, information to bear on the problem of theorem-proving search for Question-Answering <QA) Systems is descrilsed. The approach is embodied in a search algorithm, the 0* search algorithm, developed to control deductive searches in an experimental system. The Q* algorithm is part of a system, termed the Maryland Refutation Proof Procedure System (MRPPS), which incorporates both the Q* algorithm, which performs the search required to answer a ouery, and an inferential component, which performs the logical manipulations necessary to deduce a clause from one or two other clauses. The inferential component includes many refinements of resolution.

The Q* algorithm generates nodes in the search space, applying semantic and syntactic information to direct the search. The use of semantics permits paths to be terminated and fruitful paths to be explored. The paper is restricted to a description of the use of syntactic and semantic information in the Q* algorithm.

Keywords and Phrases: deductive system, heuristics, problem-solving, proof procedure system, auestion-answering, resolution, search strategies, semantic heuristics, syntactic heuristics, theorem-proving.

1. Introduction

The purpose of this paper is to describe an approach for bringing semantic, as well as syntactic, information to bear on the problem of theorem-proving search. The approach is embodied in a search algorithm, the Q* search algorithm, developed to control the search in an experimental theorem-proving-based question-answering system. This system, termed the Maryland Refutation Proof Procedure System (MRPPS), incorporates both the Q* alqorithm which performs the search required to answer a ouery, and an inferential component which performs the logical manipulations necessary to deduce a clause from one or two other clauses.

The inferential component of MRPPS provides a variety of resolution-based inference rules. For a given query, a user may select one, or a combination of the following inference systems: unrestricted binary resolution with factoring, set-of-support, input resolution, PI-deduction, A-ordering, paramodulation, linear and SL resolution. A MRPPS user may also select certain search options that are available in the system.

The clauses from the negation of a given input query and those in the MRPPS data base together with the selected inference system define a search space which is a directed graph whose nodes (states) are (labeled by) clauses. The initial states are nodes labeled bv clauses from the negation of the query, and a goal state is a node labeled by the null clause, □ .

The Q* algorithm generates nodes in the search space, applying semantic and syntactic information to direct and limit the search. This paper is restricted to a discussion of the Q* algorithm. For a detailed description of the algorithm and of the entire MRPPS system, see Minker, et al. [1972].[1] An overview of MRPPS and of its inferential component may be found in Minker, et al. [1973].[2]

The Q* algorithm has been developed for eventual use in a question-answering system of practical scale. Such a system would have a "large" number of axioms stored in its data base. A substantial majority of these axioms would be fully instantiated unit clauses, termed "data axioms." The remaining axioms, called "general axioms," define the predicates used and their interrelationships. One would want such a system to be very restrictive in permitting axioms to enter the search, in order to limit the number of I/O accesses to the data base as well as to avoid clause interaction and memory clutter. In addition, one would want to be able to answer simple questions simply. An effective theorem-proving search algorithm for a guestion-answering system should generate the search space by: (1) selecting only those axioms from the data base that are *relevant* to the query, such that the "more premising" ones enter the search first; and (2) deducing clauses from clauses already generated in as optimal an order as possible.

Both of these problems are handled in the Q* algorithm. The algorithm is subdivided into two distinct but cooperating subalgprithms: the base clause selection algorithm which handles the first problem, and the deduction algorithm which handles the second problem. The base clause algorithm uses primarily semantic information, while the deduction algorithm uses primarily syntactic information.

Before discussing the algorithm, we review some of the background which precedes the present work and upon which this work has been based.

2, Background

There has been a great deal of research in mechanical theorem-proving since the resolution principle was introduced by Robinson [1965]. Most of the research has been centered on developing refinements of resolution which reduce the size of the search space. However, relatively little work has been reported in

developing improved search strategies or using semantics for theorem-proving systems. (We have recently learned that Reiter [1972][4] is attempting to incorporate semantics with a theorem prover by the use of models.) Resolution theorem-proving programs typically use the unit preference search strategy, developed by Vtos, et al. [1964]? in which the merit of a clause in the search is based on its length. While this is a useful strategy for proving simple theorems, it is not nearly powerful enough to successfully guide a search for even moderately deep mathematical proofs. Furthermore, it is not adequate for use in question-answering applications which consist largely of unit clauses, since it gives preference to ail unit clauses without regard to their relevance to the query.

Green [1969][6] in considering the application of theorem-proving in question-answering systems, partitioned clauses into two sets, active and passive. Active clauses are inferred clauses and axioms selected from the data base, whereas passive clauses are axioms which have not yet been selected. Only active clauses may be used in inferences. A passive clause is made active only if it resolves with an existing active clause; preference is given to passive unit clauses. Green incorporated this search approach with the unit preference strategy in the QA3 system which employs, essentially, set-of-support as its inference system.

Kowalski [1970a, 1970b][7,8] Separates the notions of an inference system and a search strategy and describes their respective roles in the theorem-proving search problem. In addition, he defines a class of "upwards diagonal" search strategies which may be applied to arbitrary problem domains. These strategies simultaneously extend and refine the class of search strategies, which includes the A* algorithm described by Hart, Nilsson and Raphael [1968][9] Kowalski also presents a particular upwards diagonal search algorithm, which he calls the J algorithm. In the J* algorithm, the passive/active clause concept is employed. In addition, the algorithm employs a clause merit evaluation function in which both the length and level of a clause are considered. Upwards diagonal search strategies could also be defined using heuristics other than length and level (see Minker, et al. [1972])[1]

In the 2 algorithm, the merit f(C) of clause C is a tuple, (i,j) , in which i = length(c) , and j = level(C) . Kowalski defines an ordering on this merit function which he terms "upwards diagonal merit," denoted by the symbol u . Given clauses C_1 of merit (i, ,j_1) and C_2 of merit (i_2,j_2) tnen C_1 £_u C, (I.e., C, is of better or equal upwards diagonal merit than C_2) if

(i) $i_1 + j_1 < i_2 + j_2$, or

(ii) $i_1 + j_1 = i_2 + j_2$ and $i_1 < i_2$.

Thus, if the length and level of two clauses sum to the same value, it is better to generate the shorter clause, since the goal of the search strategy is to generate a clause of length zero. It is thus advantageous to discriminate between clauses in this manner rather than solely on the basis of the sum of the merit components, as is done in earlier search algorithms such as the A* algorithm (Hart, et al. [1968] P.

Based on this merit function, Kowalski partitions the search space into merit sets, called A-sets, into which clauses of equal merit are generated. The *J** algorithm generates clauses into these merit sets in increasing upwards diagonal order, It first tries to generate a clause into A(0,0) which would be possible only if the null clause were in the starting set of base clauses. It then successively attempts to generate (by selecting a clause from the base set, or by inference) clauses into the sets A(0,1) , A(1,0) , A(0,2) , A{1,1) , A(2,0) , A(0,3) , etc. Any time a clause is generated, an attempt is made to recursively generate all of its successors and successors of its successors which are of better merit than the current merit to which the algorithm has sequenced.

While the above description of the I^{r*} algorithm is necessarily terse, the reader is referred to Kowalski [1970a, 1970b],[7,8] and to Minker, et al. [1972, 1973][1,2] for more detailed discussions.

3. A View of Resolution Theorem-Proving Search as Problem-Reduction Search

Before describing the operation of the Q* algorithm, it will be useful to describe how resolution theorem-proving search may be interpreted as problem-reduction search. In a problem-reduction search, an operator is applied to a problem P to reduce it to a set of subproblems such that solution of all the successor subproblems implies a solution to P . Equivalently, the conjunction of these subproblems may be considered a new problem which must be solved for the given problem to be solved. This reduction process is recursively applied to each generated subproblem until the oriqinal problem has been reduced to a set of primitive subproblems which are trivial to solve.

Moreprecisely, a problem-reduction consists of three sets:

(1) a set of starting problems;
(2) a set of operators that reduce a problem to a set of subproblems; and,
(3) a set of final problems which are, by definition, solved.

In the context of resolution theorem-proving, a clause corresponds to a problem and a literal corresponds to a subproblem. Thus, a clause is a *conjunction* of subproblems since *all* subproblems must be solved for the problem to be solved (i.e., all literals must be eliminated).

The startinq set of problems corresponds to a starting set of base clauses. The actual members of this set will depend upon the inference system being used. For instance, if the inference system is set-of-support, or if it includes set-of-support, the starting set of problems corresponds to the clauses in the negation of the cruery. For other inference systems, it consists of the entire set of base clauses, i.e., all of the axioms together with the query clauses.

Clauses may also be regarded as operators which are applied to problems by means of inference rules. However, the set of operators that may be applied will vary depending on the inference rule used, e.g., factoring, paramodulation, and various refinements of resolution. For a resolution operation, two clauses are involved. In general, the choice of which clause is the problem and which is the operator is somewhat arbitrary. However, for certain inference systems, it seems natural to make a clearer distinction between the two. For instance, if input resolution is used, each clause of a linear chain may be regarded as a problem and all base clauses may be regarded as operators. For linear and SL resolution, ancestors of a problem may also be considered as operators, in addition to all the base clauses.

A clause may also be a final problem. In particular, the null clause is the only such clause when one is attempting to refute a conjecture. Thus, when a null clause is generated, we know that a starting problem has been solved. Determining when a subproblem is solved is more difficult. This may be seen by considering the process involved in solving a subproblem. A subproblem P (which corresponds to a literal in a clause), may be solved in either of two wavs: 1) it may be solved in a single step bv applving an operator that refutes it (a unit clause); 2) alternately, an operator may be applied to P that spawns a set of subproblems such that P is solved if and only if all of the subproblems are solved. In practice, the determination of when a given subproblem has been solved may be quite difficult. Depending on the inference system used, subproblems are attacked in arbitrary order. Consequently, the bookkeeping required to keep track of which literals of a given clause have been spawned by which subproblems may be quite involved.

However, one inference system, SL resolution {Kowalski and Kuehner [1971]},10 stands out as being particularly well suited for this task. In the first place, SL resolution requires that only one subproblem be attempted at a given level. Secondly, the required bookkeeping is built into the clause representation used in SL resolution. In this representation, a clause is referred to as a chain. In SL resolution, a subproblem to which a solution is sought is actually carried along (and duly tagged) in each of the successor chains. Such literals are called "A literals" in the successor clauses. As new literals (called "B literals") are introduced in the attempt to solve a subproblem, they are placed to the right of the A literal. When all such B literals have been resolved awav (resolution is only performed on rightmost B literals) and an A literal is exposed on the right, then that constitutes a solution to the subnroblem from which that A literal descended. At this time, SL resolution requires that the exposed A literal be removed from the clause - an operation which is called truncation. Only at this time, (and not before as with other refinements of resolution) may a new subproblem from the original problem be selected for solution.

4. The Q* Algorithm

The Q* algorithm is based upon the *I* algorithm of Kowalski together with the idea of Green to allow only certain base clauses to enter the active clause space. Both of these approaches have been extended in the current algorithm. The Q* algorithm is subdivided into two major components:

1) the deduction algorithm which uses primarily syntactic information; and,
2) the base clause selection algorithm which uses primarily semantic information.

The deduction algorithm generates new problems by the application of operators to problems already in the search space. Any operator that is applied must itself have been previously generated either bv inference or by the selection of a base clause.

Problems and operators are generated bv two major subalgorithms, FILL and RECURSE, which are adaptations of analogous subalgorithms of the £ algorithm. As a clause is generated, it is placed into an A-set corresponding to its merit. The algorithm uses a generalized upwards diagonal merit function to calculate the merit of a clause, and generates clauses in approximately upwards diagonal order. (Further details on this function as well as on the deduction algorithm are contained in Minker, et al. [1972, 1973][1,2].)

In using this merit function, it is hoped that clauses of better merit will be more useful to answering the query than those of worse merit. However, in practice, it is extremely difficult to develop comnonents of the merit function that adequately reflect the relevance of clauses with respect to the query. Most components used to date have been syntactic in nature, rather than semantic, and have led to very inefficient search strategies. (See Slagle and Farrell [1971][11] and Minker, et al. [1972][1] for examples.) Consequently, theorem-provers using such a search strategy are often deluged with irrelevant clauses and thus are inadequate for most applications.

There are two ways to alleviate this problem. First, after a problem is generated by an inference step, various deletion rules may be applied so that the problem may be eliminated in case it is redundant or semantically meaningless. Thus, tautologies, alphabetic variants and subsumed clauses may be eliminated. Furthermore, subproblems as well as problems mav be eliminated by predicate evaluation. This mav be done bv referencing stored semantic information about the problem domain. For instance, let C v F(Mary, x) be a generated problem, where C is the remainder of the clause. If it is known that the first argument of the father predicate must be male, and we know that Mary is female, then the literal F(Marv, x) will never have a solution (i.e., no unit clause/ F{Mary, a) , can be true, for any person a). We can thus evaluate the literal F(Mary, x) as being true relative to the interpretation given to the problem domain and may eliminate the entire problem from the search space since it is *unsolvable.* With most inference systems completeness should not be violated bv deleting the clause. On the other hand, if a literal of a clause is evaluated as false, that literal alone, mav be removed, since it corresponds to a subproblem that is *solved.*

Although predicate evaluation using semantic information about the problem domain will be useful in theorem-proving, it does not *prevent* irrelevant clauses from being generated and entered in the search snace. We thus feel that a more effective method by which to cut down the size of the search space is to *avoid generating* irrelevant clauses in the first place. In particular, this can be accomplished by carefully selecting those *operators* that are most relevant to the search in progress and by inhibiting those which are irrelevant. This is the function of the base clause selection algorithm.

Depending on the inference system, the algorithm treats one or all of the literals of a generated clause as a specification with which to select axioms. Thus, each literal is termed a "spec literal." (This mav be viewed as the selection of operators to apply to a subproblem.) In order to select axioms relevant to the querv, the Q* algorithm initially generates clauses from the negation of the query. Each spec literal is used to *locate* those axioms which either resolve with the generated clause on that literal, or, in some cases, which possess a literal which will unify with the spec literal. The axioms that have been located for a particular spec literal may be reduced in number by *filtering out* those which are found to be semantically inappropriate (e.g., because of incompatible argument types). The remaining operators for a given subproblem then become candidates for generation and must be *ordered* so that "more promising" operators are tried first. The list of candidates for a spec literal is placed on a list called the SPECLIST, which is itself ordered by using various

heuristic criteria. Thus, the ordering of the SPECLIST reflects the judgment of the base clause algorithm as to which subproblems from among all subproblems in all generated clauses should be attacked first. That is, from among all literals in all generated clauses that are eligible for resolving, the base clause selection strategy picks the best one.

As a result of the control maintained over the base clauses by the base clause selection algorithm, the FILL operation of the deduction algorithm will not, in general, acquire all base clauses of a given FILL merit. It obtains only those base clauses made available to it by the base clause selection algorithm. As a result, the base clauses that are selected will not necessarily be generated in merit order. Consenuently, and in contrast to the J* algorithm, the Q* algorithm is not admissible. However, as the exanples in Section 5 demonstrate, the loss of admissibility may be of only theoretical importance since it would seem to be more important in practical QA amplications to find anv solution quickly rather than to find a simplest solution at great expense, at the risk of finding no solution at all because of space and time considerations.

The following sections describe the manner in which axioms are located, the way certain of these are filtered out, and the way those that remain are ordered. Section 4.4 presents a discussion of a olanned extension of the search strategy in which the base clause algorithm may prune the search space and delete certain candidates from the SPBCLIST upon the recognition of certain events in the search process.

4.1 Locating Candidate Axioms

As already indicated, the literals of generated clauses are used to locate axioms which may become candidates for generation. The inference system that is being used by the deduction algorithm will determine which generated clauses should be used for this purpose, and what criterion should be applied to select the axioms. If the inference system in use is set-of-support, or if it incorporates set-of-support as does SL resolution, then only the literals of supported clauses will be used. Furthermore, the axioms which become candidates will be those which could *resolve* with the generated clause. On the other hand, if the inference system does not include set-of-support then all genera-ted clauses, including generated axioms, will be used to locate axioms. In addition, the resolution criterion must be weakened, so that an axiom will potentially become a candidate if it contains a literal which *unifies* with a literal of a generated clause. The weakening of the resolution criterion is necessary in order to assure the refutation completeness of the algorithm, e.g., in cases where the only refutations for a given ouery involves the use of a lemma, and the ouery is not used until the lemma has been established.

Not only does the inference system dictate which clauses are to be used in locating axioms, it also dictates which literals should be used. Thus, in "single literal" inference systems such as SL resolution or A-ordering, only one designated literal of each generated clause is used, while for other inference systems, all of the literals will be used for this purpose.

The approach to locating axioms which may enter into the search, may, for a given query, preclude the generation of certain axioms. That is, there may be axioms which are completely unrelated to the ouery in that they concern a different problem domain. But since these are completely irrelevant to the query, the search strategy will not be hindered from finding a refutation if one is attainable.

4.2 Semantic Filtering of Candidate Axioms

Once the potential candidates for a given spec literal are found, these may be subjected to semantic filtering, and only those which are semantically consistent with the spec literal and its host clause may become candidates. This filtering may be done on the basis of the class membership, or *type,* of a variable or constant in the spec literal. For example, suppose the generated clause literal is ~PARENT (Dan, Brett) and it is known that Dan is a male, or the literal is ~PARENT (x, Brett) and, from context, it is known that x corresponds to a male. Suppose further that the two potential base clause candidates:

(1) ~FATHER(u,v) V PARENT(u,v) , and
(2) ~MOTHER (u,v) V PARENT (u,v)

were found in the data base. Applying the filter, axiom (2) would be found semantically inconsistent with the spec literal since the variable u in the PARENT literal is found to be of type female from context, (i.e., from its use in MOTHER literal), and what is necessary is a PARENT literal whose first argument is of type male. Thus, only (1) would become a candidate.

A further type of filtering may also be performed. When all of the solutions to a subproblem are explicit in the data base, it becomes unnecessary to generate any general axioms which might be used to deduce a solution to the subproblem. For example, let ~MOTHER(x, Emily) be a generated clause literal. Also assume that it is known that every individual has exactly one mother. If the axiom MoTHER(Roz, Emily) were found in the data base, then, since this completely solves the subproblem at hand, it would become the only candidate. No other axioms that could resolve with ~MOTHER{x, Emily) would be entered to satisfy this subproblem.

The filtering we have described can be seen to reduce the number of axioms which are generated into the search space. As a result, fewer clauses will be available to interact logically and so the total number of clauses which may be generated is thereby reduced, clearly, the clauses of the implicit search space which have become ungeneratable as a result of this filtering could not have contributed to a refutation so that although the search algorithm fails to be *complete* (in the sense of exhaustive), it remains *refutation* **complete.**

4.3 Ordering of Candidate Axioms and Subproblems

Once the candidate axioms have been located and subjected to semantic filtering, they must be ordered so that the "more promising" ones will be generated first. As already noted, there is a two-level ordering that is performed: the candidates associated with a particular spec literal must be ordered (this corresponds to the ordering of operators to be applied to a subproblem), and the lists of candidates for the various spec literals are ordered with respect to one another (this corresponds to the ordering of subproblems).

In ordering the candidates for a particular spec literal, two different approaches may be employed. In one approach, they may be ordered according to a *Teooimendation* by the user. This is analogous to the recommendation lists in PLANNER (Hewitt, [1971])[12]. Another approach is to order candidates according to

the merit ordering, f , used by the deduction strategy, making sure that data axioms (i.e., fully instantiated unit clauses) precede any general axioms.

With respect to ordering the subproblems, a heuristic guideline is applied when one or more constants occur in literals of the query clauses. In this event, the candidates located by using constant-carrying literals of generated clauses will precede in the ordering those candidate axioms located by the use of general literals. The reasoning behind this heuristic is that (1) constant-carrying literals will generally resolve (or unify) with fewer base clauses than will general literals, and (2) if a query is about a particular individual (constant), then the search mechanism will make *more informed* decisions about the *relevance* of candidate axioms if it concentrates on this individual, -and to others which are found to be related to it, rather than going off blindly on the basis of some general literal. In many cases, the general literal may actually unify with a large subset of the axioms in the data base while only a few of these may be relevant. Furthermore, by pursuing this policy, literals which are uninstantiated at one level of the search may become instantiated at a subsequent level. Thus, in this approach, it will often be the case that clues (in the form of constants) to direct the search will be passed from one subproblem to another as the search progresses.

Figure 1 illustrates a proof that was derived giving preference to those literals that contain constants. In Figure 1, clause (l)yields two literals for the SPECLIST. The literal M(y, Sally) would be given preference on this list since it contains a constant. The axioms that can interact with the literal M(y, Sally) are entered into the search before other axioms that may apply to that_clause. Similarly, in clause (3), the constant in H(v, Rita) gives this literal preference. Although proofs can be found without using this heuristic, such proofs will tend to generate many unneeded clauses.

It is reasonable to assume that in question-answering applications a large percentage of the queries will contain constants. Even if constants appear in a given literal, it may be reasonable to give preference to a literal containing only variables based upon a lower estimate of potential candidates satisfying each of these literals. However, we would expect this to happen infrequently in question-answering systems.

There may be several subproblems (spec literals) containing constants which must be ordered with respect to one another on the SPECLIST. In this case, the ordering of candidate lists by their spec literals is accomplished by using a *prediction* of the merit, f , of a resolvent between the spec literal's host clause and the first axiom in the ordered list of candidates from that literal. One consequence of this ordering is that if a generated unit clause is contradicted by a unit axiom, then this axiom surely will have become a candidate, and it will be placed first in the ordering so that it will be the very next axiom to be generated. Each time an axiom is removed from the list of candidates, the next candidate in the list is used for a new prediction, and the corresponding SPBCLIST entry is reinserted into an appropriate position of the list — it may remain on top of the list, or some other subproblem may emerge as "most promising."

In some data bases, there may be certain constants which occur in a large number of the data axioms. Generally, these would be non-specific, class specifying types of constants such as MALE and SINGLE as used in the data axiom.

PERSON(JOHN, SMITH, ID475, MALE, FEMALE) .

Since such constants do not designate particular individuals, and since they do occur throughout the data base, thev will be of little value in directing the search. Thus, for the purpose of ordering the SPECLIST, spec literals containing these types of constants alone are treated as general literals.

4.4 Semantic Actions During the Search

It was noted in Section 4.2 that some subproblems may be comnletelv solved by data axioms stored in the data base, More generally, some subproblems have an exact number of solutions while others have an indefinite number of solutions. If the number of solutions to a given subproblem is known, then, during the course of the search, the progress of the search relative to finding these solutions can be monitored. When all of the solutions have been found, that portion of the search graph which has been unnecessarily generated in the attempt to find these solutions can be pruned. In addition, the candidate axioms located through the use of literals in the pruned clauses or in clauses along the solution path will have their candidate status removed. The effect of this pruning is to reduce the number of irrelevant clauses in the search space. Of even greater significance is that the potential successors of these clauses are effectively prevented from being generated.

The bookkeeping required to apply these semantic actions would be quite complex when arbitrary inference systems are employed, since, for a given clause (problem), attempts to solve all of the subproblons are carried out simultaneously. However, SL resolution is particularly well suited for this approach, since only one subproblem is attacked at a time and because the bookkeeping required to detect the solution of a subproblem is built into the SL clause representation, as noted in Section 3. Semantic actions are taken when truncation occurs, since truncation only occurs when a subproblem has been solved. The semantic action taken is to increment a count and test it against the number of solutions sought. If this number is met, then pruning can be performed.

The above approach does not require that all solutions to a given subproblem be obtained before progressing to another subproblem. It would seem to be desirable, in some cases, to continue the search for additional solutions to the subproblem, while at the same time advancing to the next subproblem.

5. Example Runs With MRPPS

The MRPPS system incorporates a preliminary version of the Q* algorithm plus an option for selecting the algorithm. The current implementation of Q* does not yet include the semantic filtering of candidates, and no mechanism has been included to take semantic actions. The current version includes the deduction strategy as outlined, together with a base clause selection strategy which locates and orders axioms essentially as described. All details concerning the current implementation are described in Minker, et al. [1972][1]. The implementation of the £ algorithm uses the same deduction strategy as Q*, but the base clause selection strategy merely locates axioms in merit order, that is to say, by length, without regard to other clauses generated by the search.

We present comparative statistics for two different queries, in which set-of-support and SL resolution were the chosen inference systems, and the merit components were clause length and level. Clause length and level bounds were both set at eleven, and an f value bound (= length + level) was set at eleven. The data base used contains 295 data axioms which explicitly state the (1) mother, (2) father, (3) birth date, and (4) birthplace of individuals in terms of their identification (i.d.) numbers, and (5) which relate the first and last name of each individual to their identification number. The data base also includes 88 general axioms which define the various predicates in terms of other predicates.

Question 1. "What is the name of Brett Fishman's moth-

The significant aspects of this simple question is that its negation in clause form is a unit clause containing constants. With the data base used, it requires four axioms and four levels of search to achieve a refutation.

Both the *v** and Q* algorithms successfully completed the search, however as the statistics in Table 2 demonstrate, with considerably different levels of work and resources required. Whereas the Q* algorithm fortuitously generated the minimal number of base clauses, the *}** algorithm has to generate all axioms of length one (i.e., all the data axioms), two, and three and some axioms of length four before the required general axiom (of length four) was obtained. Once this general axiom was resolved against the query clause, the resolvent, having support, could be resolved against many of the axioms already in A-sets, and similarly for the resolvents obtained. As a consequence, many more resolvents were generated. Furthermore, the presence of a large number of clauses in A-sets required the use of a considerable amount of free space to keep track of the clauses as well as a significant increase in the search time required.

Question 2. "Is there a person who is the grandfather on the father's side of individual 67 and the grandfather on the mother's side of individual 7?"

The negation of the Question in clause form is:

FF(x, 67) V FM(x, 7)

where FF is the predicate "grandfather on the father's side", and FM is the predicate "grandfather on the mother's side." This question, which requires six levels of search requires the following six axioms from the set of 295 data axioms and 88 general axioms:

F(x, y) V M(y, z) v FM(x, z)

F(x, y) V F{y, z) V FF(x, z)

M(4, 7), F(6, 4), F(6, 63), and F(63, 67) .

Proofs were found using set-of-support and SL resolution as the inference mechanism. The statistics for set-of-support indicate that the Q* algorithm was quite selective as compared to both with respect to the number of resolvents and axioms generated. However, 0* did obtain considerably more general axioms than actually needed.

The same question was tested using SL resolution as the inference mechanism. The table shows statistics for the Q* algorithm as generated by the computer, it also shows statistics, generated by hand, for the Q* algorithm when the semantic rule of counting the number of possible solutions to a subproblem is used. The Q* and *1* algorithms without the use of semantics generated the same number of inferences. However, £ brought in 381 axioms, only 6 of which were needed. Q* when not using semantics brought in 16 axioms, only 6 of which were actually needed. Q* without semantics performed better when using SL resolution than when using set-of-support *as* the inference mechanism. The use of semantics for Q* shows its general utility. The number of axioms entered were 7, while only 6 were needed for the proof. Although there is a significant decrease in the number of axioms generated using semantics, the one example should not be considered to be the type of improvement that one will always achieve using semantic considerations to restrict the search. However, it is indicative of the use of semantics. The trace of clauses generated by SL resolution and the proof are shown in Figure 2.

6. Summary

We have described a search strategy for theorem-oroving which uses both svntactic and semantic information to direct the search in a Question-Answering System. Most theorem-proving systems have used primarily syntactic information such as clause level and clause length to direct the search. Others have attempted to use only refinements of resolution with a breadth-first search or an ad hoc use of heuristic rules. Experience with such systems may have led some to speculate that means other than theorem-proving must be developed to perform deduction in question-answering, or in other problems (e.g., Anderson and Hayes [1972])[13] The reason for this skepticism of theorem-proving may stem from the inadequacy of search strategies to limit the search to a "reasonable" number of nodes in finding proofs. We agree that the use of only logical rules and syntactic heuristics will not be sufficient for most systems. However, it is our hope that the introduction of semantic information to aid in directing the search will be useful.

We have tried to show where semantic information may be used in a theorem prover, and how it may coordinate with the syntactic heuristics and logical deduction. There are several places where semantics enter into the search process:

(1) at the time a clause is generated, to determine whether it is semantically meaningful;

(2) in selecting a literal of a clause to expand;

(3) in the filtering out of irrelevant operators; and

(4) in taking actions when a literal is solved.

There is, however, a tradeoff as to when semantics should be applied. It may result that the attempt to determine whether or not a clause is solvable could take more time than the solving of the problem. Hence, studies must be performed in this area to determine the various costs involved.

With an arbitrary inference system it is not always easy to determine when a literal from some clause has been solved. The bookkeeping that might be involved could become unmanageable. However, as described previously, SL resolution minimizes the bookkeeping problem. SL resolution has the additional advantage of providing the opportunity to select which subproblem to attack next.

Some of the semantic information we use is general, e.g., the count of the number of solutions to a subproblem; if such information is supplied by the

user it could be used to advantage for any problem domain. A general framework for representing and applying semantic information is needed. In addition, our semantic considerations have been directed primarily at the meaning of constants, and not at the meaning of functions and Skolem functions that may frequently occur in clauses. These must also be considered.

We have presented a first version of the Q* algorithm that carbines syntactic and semantic considerations. Improvements must be made in the algorithm to handle parallel searching such as in CONNIVER (Sussman and McDermott[1972])[14] and to implement some of the features that were described, but not incorporated into the first version. However, based on limited experience with the current version of Q*, it has been observed that the search does restrict the generation of clauses.

The area of semantics and theorem-proving requires considerably more research and experimentation. This paper has described a first step in exploring this area.

References

1. Minker, J., Fishman, D.H., and McSkimin, J.R. The Maryland Refutation Proof Procedure System (MKPPS). TR-208, Computer Science Center, University of Maryland, College Park, Md., 1972.

2. Minker, J., McSkimin, J.R., Fishman, D.H. MRPPS-An Interactive Refutation Proof procedure System for Question-Answering. TR-228, Computer Science Center, University of Maryland, College Park, Md., 1973.

3. Robinson, J.A. "A Machine Oriented Logic Based on the Resolution Principle." J.ACM 12, l(Jan. 1965), 23-41.

4. Reiter, R. The Use of Models in Automatic Theorem Proving, Technical Report 72-09, Department of' ' Computer Science, University of British Columbia, Vancouver B.C., Canada, September, 1972.

5. was, L.T., Carson, D.F., and Robinson, G.A. "The Unit Preference Strategy in Theorem Proving." Proc. FJCC, Thompson Book Co., New York, 1964, 6X5-621.

6. Green, *C.C.* "Theorem Proving by Resolution as a Basis for Question-Answeriiig Systems." In: Meltzer, B. and Michie, D. (Eds.), Machine Intelligence 4, American Elsevier, New York, 1969, 183-205:—

7. Kowalski, R. Studies in the Completeness and Efficiency of Theorem-Proving by Resolution. Ph.D. Thesis, U. of Edinburgh, 1970a.

8. Kowalski, R. "Search Strategies for Theorem Proving." In: Meltzer, B. and Michie, D. (Eds.), Machine Intelligence 5, American Elsevier, Hew York, I970b, 181-266.

9. Hart, P., Nilsson, N., and Raphael, B. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths." IEEE Trans. Sys. Sci. Cybernetics 4, 2(1968), 100-107.

10. Kowalski, R. and Kuehner, D. "Linear Resolution with Selection Function." Artificial Intelligence 2, 3/4, (1971), 221-260.

11. Slagle, J.R. and Farrell, CD. "Experiments in Automatic Learning for a Multipurpose Heuristic Program." Ccmm. *ACM* 14, 2(Feb. 1971), 91-98.

12. Hewitt, C. "Procedural Embedding of Knowledge in PLANNER." Proc. IJCAI, British Computer Society, London, England, 1971, 167-182.

13. Anderson, D. and Hayes, P.J. An Arraignment of Theorem-Proving or the Logician's Folly, Memo No. 54, Dept. of Computational Logic, School of A.I., University of Edinburgh, Scotland, 1972 .

14. Sussman, G.J., and McDermott, D.V. "From PLANNER to CONNIVER - A Genetic Approach." Proc. FJCC, AFIPS Press, Montvale, N.J., 1972, 1171-1179.

	Question 1		Question 2				
	Q^*	Σ^*	Q^*	Σ^*	Q^*		Σ^*
					No Semantics	Semantics (2)	
# inferred clauses	11	65	39	492	8	6	8
# Factors	1	1	3	3	0	0	0
# resolvents	10	64	36	489	8	6	8
# axioms generated (1)	4	342	18	381	16	7	381
# data axioms	3	295	6	295	5	4	295
# general axioms	1	47	12	86	11	3	86
# axioms needed	4	4	6	6	6	6	6
# data axioms	3	3	4	4	4	4	4
# general axioms	1	1	2	2	2	2	2
# variants eliminated	3	17	8	122	0	0	0
# clauses in A-sets	13	391	50	752	25	14	390
freespace used (words) (5)	431	9464	1303	43751	375	~267	1137
time required (seconds)	13	12.4	1.8	59.0	.37	<.37	2.5
Inference System	SOS (3)	SOS	SOS	SOS	SL (4)	SL	SL

Table 1

(1) Figures do not include the clause from the negation of the theorem

(2) These calculations were performed by hand since the routines that are to handle semantics have not yet been implemented.

(3) Set-of-support

(4) Linear resolution with selection function

(5) Freespace refers to the storage needed by clauses formed by resolving or factoring

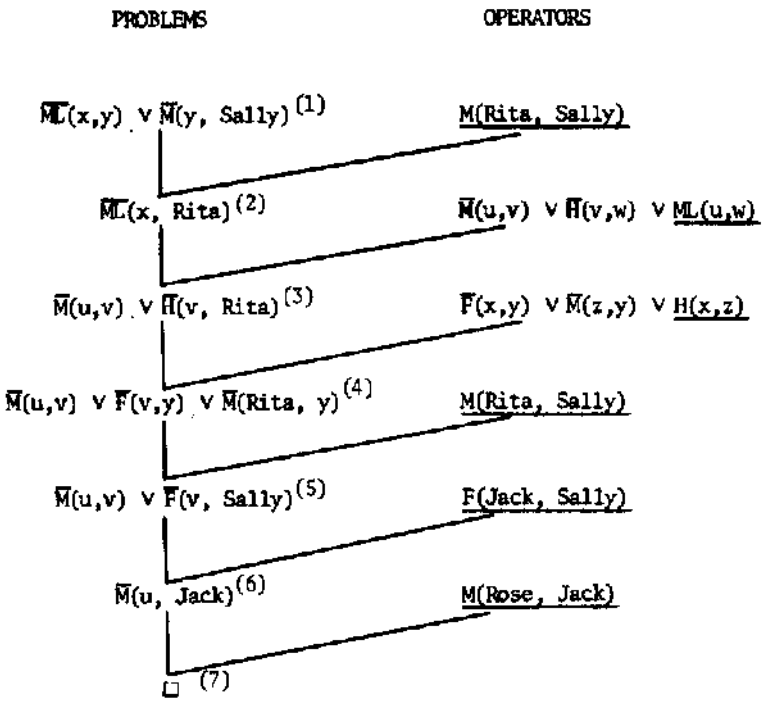

Figure 1. The use of constants to guide the search.

TRACE OF GENERATED CLAUSES

1 (-THEOREM) -FF(x3,67)+-FM(x3,7)

2 (AXIOM) -F(x,y)+-M(y,z)+FM(x,z)

3 (1, 2) -FF(x3),67)+[-FM(x3,7)]+-F(x3,x4)+-M(x4,7)

4 (AXIOM) M(4,7)

5 (3, 4) -FF(x3,67)+[-FM(x3,7)]+-F(x3,4)+[-M(4,7)]

6 (5, 5) -FF(x3,67)+[-FM(x3,7)]+-F(x3,4)

7 (AXIOM) F(6,4)

8 (6, 7) -FF(6,67)+[-FM(6,7)]+[-F(6,4)]

9 (8, 8) -FF(6,67)+[-FM(6,7)]

10 (9, 9) -FF(6,67)

(11) (AXIOM) -FMA(x1,x2,y1,y2)+-NAM(x1,x2,x,u,v,w)
+-NAM(y1,y2,y,v1,v2,v3)+FM(x,y)

12 (AXIOM) -F(x,y)+-F(y,z)+FF(x,z)

(13) (AXIOM) -M(x,y)+-W(x,z)+F(z,y)

(14) (AXIOM) -NAM(x1,x2,y,M,u1,u2)+-O(x,y)+F(y,x)

(15) (AXIOM) -F(x,y)+-H(x,z)+M(z,y)

(16) (AXIOM) -NAM(x1,x2,y,F,u1,u2)+-O(x,y)+M(y,x)

17 (AXIOM) -FFA(x1,x2,y1,y2)+-NAM(x1,x2,x,u,v,w)
+-NAM(y1,y2,y,v1,v2,v3)+FF(x,y)

(18) (AXIOM) -FA(x1,x2,y1,y2)+-NAM(x1,x2,x,u,v,w)
+-NAM(y1,y2,y,v1,v2,v3)+F(x,y)

(19) (AXIOM) -MA(x1,x2,y1,y2)+-NAM(x1,x2,x,u,v,w)
+-NAM(y1,y2,y,v1,v2,v3)+M(x,y)

(20) (1,11) -FF(x3,67)+[-FM(x3,7)]+-NAM(x1,x2,x3,u,v,w)
+-FMA(x1,x2,y1,y2)+-NAM(y1,y2,7,v1,v2,v3)

(21) (AXIOM) NAM(MJA,HKA,7,M,1,28)

(22) (20,21) -FF(x3,67)+[-FM(x3,7)]+-FMA(x1,x2,MJA,HKA)
+-NAM(x1,x2,x3,u,v,w)+[-NAM(MJA,HKA,7,M,1,28)]

(23) (22,22) -FF(x3,67)+[-FM(x3,7)]+-NAM(x5,x4,x3,x8,x7,x6)
+-FMA(x5,x4,MJA,HKA)

(24) (AXIOM) -FM(x,y)+-NAM(x1,x2,x,u,v,w)+-NAM(y1,y2,y,v1,v2,v3)
+FMA(x1,x2,y1,y2)

25 (10,12) [-FF(6,67)]+-F(6,y)+-F(y,67)

26 (AXIOM) F(63,67)

27 (25,26) [-FF(6,67)]+-F(6,63)+[-F(63,67)]

28 (27,27) [-FF(6,67)]+-F(6,63)

29 (AXIOM) F(6,63)

30 (28,29) [-FF(6,67)]+[-F(6,63)]

31 (30,30) [-FF(6,67)]

32 (31,31) NULL CLAUSE

PROOF

1 (-THEOREM) -FF(x3,67)+-FM(x3,7)

2 (AXIOM) -F(x,y)+-M(y,z)+FM(x,z)

3 (1,2) -FF(x3,67)+[-FM(x3,7)]
+-F(x3,x4)+-M(x4,7)

4 (AXIOM) M(4,7)

5 (3,4) -FF(x3,67)+[-FM(x3,7)]
+-F(x3,4)+[-M(4,7)]

6 (5,5) -FF(x3,67)+[-FM(x3,7)]
+-F(x3,4)

7 (AXIOM) F(6,4)

8 (6,7) -FF(6,67)+[-FM(6,7)]
+[-F(6,4)]

9 (8,8) -FF(6,67)+[-FM(6,7)]

10 (9,9) -FF(6,67)

11 (AXIOM) -F(x,y)+-F(y,z)+FF(x,z)

12 (10,11) [-FF(6,67)]+-F(6,y)+-F(y,67)

13 (AXIOM) F(63,67)

14 (12,13) [-FF(6,67)]+-F(6,63)
+[-F(63,67)]

15 (14,14) [-FF(6,67)]+-F(6,63)

16 (AXIOM) F(6,63)

17 (15,16) [-FF(6,67)]+[-F(6,63)]

18 (17,17) [-FF(6,67)]

19 (18,18) NULL CLAUSE

LEGEND

Trace of Generated Clauses - No Semantics Used
Clauses 1-32 generated.

Trace of Generated Clauses - Semantics Added
All numbered clauses that are not circled were generated.

Interpretation of Predicate Letters -

F - father
M - mother
NAM - name (correlates alphabetic name and number of person)
FF - father of the father (grandfather on the father's side)
FM - father of the mother (grandfather on the mother's side)
FMA - alphabetic name of the maternal grandfather
FA - alphabetic name of the father
MA - alphabetic name of the mother
W - wife
FFA - alphabetic name of the paternal grandfather
O - offspring

Syntax of Chain -

[ ] denotes an A literal - clauses truncated one literal at a time

\+ denotes disjunction (V)

Figure 2. Computer Output for Question 2 With SL Resolution