Report 83-21
Stanford -- KSL

Finding All of the Solutions to a Problem.
David E. Smith, Michael R. Genesereth,
Sep 1983

Scientific DataLink

Stanford Heuristic Programming Project
Memo HPP-83-21

September 1983

Finding All of the Solutions to a Problem

David E. Smith

Michael R. Genesereth

COMPUTER SCIENCE DEPARTMENT
Stanford University
Stanford, California 94305

Table of Contents

1. Introduction 2

2. Basic Architecture 4

3. Knowledge About the Number of Answers 5

3.1 The Language 5

3.2 Knowledge About Set Sizes 5

3.2.1 Gathering Knowledge about Set Sizes Dynamically 7

3.2.2 Maintaining the Accuracy of Cardinality Information 8

3.2.3 Invariance 10

4. Interaction with other Meta-level Reasoning 11

5. Results 14

6. Discussion 15

6.1 The Closed World Assumption 15

6.2 Adding Other Meta-Level Reasoning 15

6.3 Perspective 16

Abstract

This paper describes a method of cutting off reasoning when all of the answers to a problem have been found. Briefly, the method involves keeping and maintaining information about the sizes of important sets, and using this information to determine when all of the answers to a problem have been found. We show how this information can be dynamically calculated and kept accurate in a changing world. Additional complexity is encountered when this maintenance is mixed with independent meta-level reasoning for pruning search spaces. Our experiences are reported for the use of these techniques in two expert system projects.

1. Introduction

Suppose that you were asked to find the names of the parents for a well known person such as Jerry Brown. Perhaps you already know that Edmund Brown, Sr. is Jerry's father (and by implication, one of Jerry's parents). Some additional digging in a reference book would determine Jerry's mother, and hence another of Jerry's parents. But why stop here? Perhaps digging harder in the library would turn up a few more of Jerry's parents.

But a person has only two parents (a mother and a father), so once they have been found, we can stop looking. This characteristic, of *knowing when all of the answers have been found*, is something that people are very good at. In contrast, current computer problem solvers are quite deficient in this regard. Either the system relies on an assumption that all of the answers are explicitly stored in the data base (or will be found by the same inference path), or it must search exhaustively to guarantee that all the answers will be found. There are many situations for which these alternatives are not acceptable. If the search space is very large then it may not be practical to search the space exhaustively. Likewise, there are many problems where the answers are not all available in the data base, and where many distinct answers result from pursuing different lines of reasoning (i.e.: different paths through the search space). As an example, in the problem above, one of Jerry's parents was immediately available from memory (in the data base, so to speak), but the second was obtained by consulting a reference book. Similarly, one might query an electronics system to find out which transistors in a circuit were not operating in the active region. Some are not active because they are "saturated". Others are not active because they are "off". Some of these are "off" because the base-emitter junction is reverse biased. Some are off because the collector is low or reverse biased. In both these examples, answers must be gathered by pursuing more than one line of reasoning.

While problems of this sort often appear as top level queries to expert systems, they also frequently arise as subgoals in satisfying other queries. For example, any query requesting a minimum or maximum value involves a search through all of the solutions to the problem. Equally troublesome, but less obvious, is when a conjunction is encountered as a subproblem of some (apparently innocuous) query. All of the solutions to one conjunct may have to be found and plugged into the remaining conjuncts in order to verify that there is no solution to that set of conjuncts.

We have encountered problems with these characteristics in the construction of several expert systems: a system for modelling renal physiology [13], a miniature tax consultant [9], and an intelligent front end for computer systems [7]. What is required in all of these examples is the ability to reason about the number of solutions to problems and to shut off the reasoning process when all of the solutions to a problem have been found.

In section 2 a basic problem solving architecture is introduced. It makes use of knowledge about the number of solutions to problems in order to halt reasoning. Details of specifying, deriving, and maintaining this knowledge are covered in section 3. The subject of section 4 is the rather surprising difficulty that arises when independent meta-level reasoning is used to prune the search space for a problem. Preliminary results are discussed in section 5 and relationship to other work is discussed in the final section.

2. Basic Architecture

Let us suppose, for the moment, that a problem solving system has knowledge about the number of solutions to any problem it might be given. Given such information it is a relatively simple matter to modify a typical system to take advantage of the information. Normally a problem solver (attempting to find all of the solutions to a problem) charges blindly ahead, discovering solutions and collecting or reporting them. It stops when it runs out of possible inferences. Instead, our "smart" problem solver would first ask the meta-level question "how many solutions does this problem have?" Then, while solving the problem, it keeps a count of the number of solutions found. Whenever this count reaches the total number expected, the problem solver can stop. A flow chart of this procedure appears in figure 2-1.

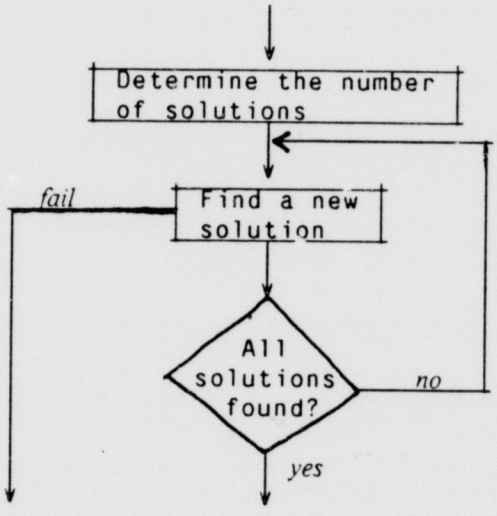

Figure 2-1: Diagram of a problem solver

This scheme is simple enough. What is necessary then, is to specify how knowledge about the number of solutions is represented, and under what circumstances a problem solver can expect to have that knowledge, or can easily infer it.

3. Knowledge About the Number of Answers

In English it is simple enough to state that a person has only two parents, or that a quadratic equation has two solutions. But to express this knowledge in a form suitable for use by a problem solver, a precise language is necessary.

3.1 The Language

Base-level and meta-level propositions will be expressed in the language of predicate calculus, but the techniques described here are not dependent on this choice. Any other language with sufficient expressive power would do as well. Several syntactic conventions are used to simplify the examples. Upper case letters are used exclusively for constants, functions, and relations. Lower case letters are used for variables. All free variables are universally quantified. Braces are used to denote sets (e.g. `{1,3,5}`) and angle brackets are used to denote ordered tuples of objects, (e.g. `<1,3,5>`).

When it is necessary to refer to a base-level expression, it will be enclosed in quotation marks, e.g. `PROVABLE('FATHER(A,B)')`. Lower case Greek letters occurring within quotation marks can be assumed to be meta-variables, i.e. they range over expressions in the base-level language. From `PROVABLE('FATHER(`ξ`,`ψ`)')`, it is legal to infer `PROVABLE('FATHER(A,B)')`.[1]

3.2 Knowledge About Set Sizes

A query to a problem solver (in which all the answers are desired) can be characterized in general as:

find all values for the variables `v` *that satisfy the proposition* `p`.

Given queries of this form, the first step for the problem solver is to:

find `n` *such that* `n=NUMBEROF(v,p)`.

where `NUMBEROF` is a meta-level function symbol that denotes the number of solutions to a problem.[2] Using this elementary vocabulary it is easy to state that the problem of finding Jerry Brown's parents has only two solutions:

```
NUMBEROF('y','PARENT(JERRY,y)') = 2
```
(3-1)

or more generally that all such problems have only two solutions:

```
NUMBEROF('y','PARENT(ξ,y)') = 2
```
(3-2)

---

[1] The approach we have adopted for representing meta-level propositions is often known as the *syntactic* approach. Readers interested in the arguments for and against this approach are referred to [12] and [14].

[2] `NUMBEROF` is referentially opaque.

If this information were provided to our problem solver, it would be able to stop reasoning after finding both of Jerry's parents. It would be a nuisance, however, if we had to specify this meta-knowledge directly for every problem a system might encounter. Fortunately, such information can be derived from simple base-level knowledge about the sizes of sets. If the members of a set correspond to the solutions of a problem, and the cardinality of the set is known, then the number of solutions to the problem will be the same as the cardinality of the set. A formal statement of this "connection" axiom is:

$$\exists\sigma(\mathrm{TRUE}(\text{`}\upsilon\in\sigma \Leftrightarrow \pi\text{'}) \;\&\; \mathrm{CARDINALITY}(\sigma)=c) \Leftrightarrow \mathrm{NUMBEROF}(\upsilon,\pi)=c. \qquad (3\text{-}3)$$

Equation (3-4) shows several examples of base-level cardinality information. Using such information, and the connection axiom given above our simple problem solver can easily infer the number of solutions to each of the corresponding problems, enabling it to halt inference when all of the solutions have been found.

$$\mathrm{CARDINALITY}(\mathrm{PARENTS}(x)) = 2 \qquad y\in\mathrm{PARENTS}(x) \Leftrightarrow \mathrm{PARENT}(x,y). \qquad (3\text{-}4)$$

$$\mathrm{CARDINALITY}(\mathrm{HEARTCHAMBERS})=4 \qquad x\in\mathrm{HEARTCHAMBERS} \Leftrightarrow \mathrm{CHAMBER}(x,\mathrm{HEART})$$

$$\mathrm{CARDINALITY}(\mathrm{SHIPSINBOSTON})=14 \qquad x\in\mathrm{SHIPSINBOSTON} \Leftrightarrow \mathrm{SHIP}(x) \;\&\; \exists d(\mathrm{DOCK}(x,d) \;\&\; \mathrm{LOC}(d,\mathrm{BOSTON}))$$

It is important to note that using plural relationships and functions, like PARENTS instead of PARENT, would not alleviate the need for having information about the size of a set. For example, knowing that:

$$\mathrm{EDMUND}\in\mathrm{PARENTS}(\mathrm{JERRY}) \;\&\; \mathrm{PRISCILLA}\in\mathrm{PARENTS}(\mathrm{JERRY}) \qquad (3\text{-}5)$$

is not enough. It is also necessary to have the information that Jerry doesn't have any other parents, a statement isomorphic to the proposition about the size of the set. Such statements are often left implicit in propositions like:

$$\mathrm{PARENTS}(\mathrm{JERRY}) = \{\mathrm{EDMUND},\mathrm{PRISCILLA}\} \qquad (3\text{-}6)$$

As a final note, there are two special cases where information about the number of solutions to a problem can be derived without domain specific information about set sizes. If the problem is to find out whether or not a proposition is true, and the requestor doesn't care about any of the variable bindings, then there is (at most) one solution to the problem. A common case of this is when the proposition p is a ground clause, i.e., contains no variables.

$$\mathrm{NUMBEROF}(\langle\rangle,p) \leq 1. \qquad (3\text{-}7)$$

Functional expressions also have at most one solution. This can be expressed as:

$$\mathrm{GROUND}(\varphi) \;\&\; \mathrm{VARIABLE}(\zeta) \Rightarrow \mathrm{NUMBEROF}(\zeta,\text{`}\varphi=\zeta\text{'}) \leq 1 \qquad (3\text{-}8)$$

3.2.1 Gathering Knowledge about Set Sizes Dynamically

There are many instances in which the addition of a priori knowledge about the sizes of sets is sufficient to solve the problems in the domain of interest. For example, in a system modelling human physiology, the number of heart chambers, the number of bones in the hand, and the number of major arteries are all unlikely to change. There are also many circumstances in which it is either inconvenient or impossible to supply that knowledge. It would be unreasonable to require that a system builder supply the cardinality of several thousand sets, each having thousands of members. Consider, for example, the number of employees for each of the departments in a large corporation. It would be more than inconvenient if this counting had to be done monthly, weekly, or daily as the systems data base was upgraded.

The situation is worse when a problem solver is dealing with a constantly changing world. Consider an expert system that serves as an intelligent interface for an operating system. The *Intelligent Agent* must accept requests from the user, make appropriate plans for their realization, perform the necessary system specific actions, and report results to the user. Knowledge about the number of tape drives and printers attached to the machine is (relatively) static and can be specified a priori. But the number of files in a given directory changes frequently and cannot be specified beforehand.

In this volatile environment there is still much that can be done. For problems where the cost of solution is high and the problem is often repeated, it is worthwhile to *cache* information about set sizes. To illustrate the technique, suppose that an *intelligent agent* has the task of finding all of the SCRIBE files on some particular directory (perhaps as a subproblem of transporting them to a new machine). The query is:

find all f *such that* FORMAT(f,SCRIBE).

For purposes of illustration, assume the intelligent agent believes that a file contains SCRIBE text if the file's extension is MSS, or if the file contains an "@make" statement at the beginning. This is stated formally as:

$$\text{NAME}(f,n) \;\&\; \text{EXTENSION}(n,\text{MSS}) \Rightarrow \text{FORMAT}(f,\text{SCRIBE}) \qquad (3\text{-}9)$$

$$\text{CONTENTS}(f,c) \;\&\; \text{WORD}(1,c,@\text{MAKE}) \Rightarrow \text{FORMAT}(f,\text{SCRIBE}).$$

In this case it is not reasonable for the system to have a priori knowledge about the number of SCRIBE files in a particular directory. It would, however, be simple enough for the system to *cache* this information after the problem has been solved once. Suppose then, that such a query is posed to the intelligent agent, and it exhaustively hunts through all of the files in the directory. Five SCRIBE files are found, three with the file extension MSS and two which do not have the usual extension, but contain "@make" statements at the beginning. After counting all of the answers, the problem solver can cache the knowledge that there are exactly five such SCRIBE files (and hence five answers to the problem):

$$\text{CARDINALITY}(\text{SCRIBEFILES}) = 5 \qquad x \in \text{SCRIBEFILES} \Leftrightarrow \text{FORMAT}(f,\text{SCRIBE}) \qquad (3\text{-}10)$$

If the query is posed again, (perhaps as a subgoal of some other request) then, according to the simple problem solving scheme of section 2, the cached information about the number of SCRIBE files (3-10) will stop the problem solving process once those five SCRIBE files are found. If the answers to the problem are also cached, then the solutions to any subsequent query would be found immediately (by looking in the data base). In this case the problem solver would not have to dissect a single filename or look at the contents of a single file. Counting the number of solutions to a problem, inferring the size of the set, and caching this knowledge permits a quicker response to future queries, just as the presence of a priori knowledge about set size would.

As we suggested in the previous section, instead of caching the cardinality of the set of SCRIBE files, it is also possible to cache the fact that each of the five files is a member of the set of SCRIBE files, and that there are no other such files. The difficulties of keeping this knowledge accurate (the subject of the next section) are the same for either of these representations.

3.2.2 Maintaining the Accuracy of Cardinality Information

There is only one problem with the scheme of section 3.2.1: succeeding requests to the intelligent agent might cause files to be added, deleted or modified. Cached knowledge about set sizes could therefore become invalid, just as cached solutions might become invalid. A standard solution to this dilemma is to keep justifications for cached information, and use *truth maintenance* [5] to remove assertions that become invalid as a result of other actions. The same technique can be used to keep track of cached knowledge about set sizes, provided that justifications are kept for these statements. In the example of the previous section, the statement "there are five SCRIBE files" rested on the assertions that there are three files with the extension MSS, and two files that begin with "@make" statements.[3] These statements, and their justifications can be recorded formally:

---

[3] There are other dependencies as well, but these are the two important ones for most cases. The view we have presented is quite a simplified one, though it is entirely possible to implement a system that performs in this manner. In reality, the cardinality statement for a set is derived from the meta-level information about the number of solutions using proposition (3-3). This meta-level proposition, in turn, has many dependencies. Among them is the statement that there are no more possible inference paths that might contribute anything to the solution of this problem.

In general, any meta-level statement about the kinds of things in the data base may be invalidated by updates to the database. This is true because such statements depend implicitly on facts of the form INDATABASE(p) or ¬INDATABASE(p). In general, when p is added to or removed from a systems database, the meta-level must be informed that INDATABASE(p) or ¬INDATABASE(p) is now true, and be permitted to make the appropriate inferences.

For a thorough analysis of the justification tree for such a statement see [17].

```
CARDINALITY(SCRIBEFILES) = 5                                                    (3-11)

JUSTIFICATION('CARDINALITY(SCRIBEFILES)=5',
              {'CARDINALITY(MSSFILES)=3','CARDINALITY(@MAKE-FILES)=2'})

CARDINALITY(MSSFILES) = 3

JUSTIFICATION('CARDINALITY(MSSFILES)=3', {'CARDINALITY(FILE-NAME-PAIRS)=237'})

CARDINALITY(FILE-NAME-PAIRS) = 237

CARDINALITY(@MAKE-FILES) = 2

JUSTIFICATION('CARDINALITY(@MAKE-FILES)=2', {'CARDINALITY(FILE-CONTENTS-PAIRS)=237'})

CARDINALITY(FILE-CONTENTS-PAIRS) = 237
```

Given these justifications, knowledge about set sizes can be kept accurate when the data base is changed. For example, suppose that the intelligent agent is instructed to delete the file named BOOK.MSS. Assuming that the intelligent agent was caching solutions as well as information about set size, its data base might contain:

```
NAME(FILE37,BOOK.MSS)                                                           (3-12)

EXTENSION(BOOK.MSS,MSS)

FORMAT(FILE37,SCRIBE)

JUSTIFICATION('FORMAT(FILE37,SCRIBE)'
              {'NAME(FILE37,BOOK.MSS)','EXTENSION(BOOK.MSS,MSS)'})
```

After performing the deletion the intelligent agent must update its data base to reflect the effects of the action. The fact "NAME(FILE37,BOOK.MSS)" is removed, and then, using truth maintenance, the fact "FORMAT(FILE37,SCRIBE)" is removed since it *depends* upon the former. In addition, the following inference must be made: since the fact NAME(FILE37,BOOK.MSS) has been removed, the fact about the number of file-name pairs is no longer valid and must be removed. By removing this cached cardinality assertion, the entire tree of cached cardinality knowledge unravels, and the statement that there are five SCRIBE files will be removed.

In general, all that is required (in addition to saving justifications and performing truth maintenance) is the inference that *a cached statement about the size of a set must be removed any time a proposition matching (unifying with) the set definition is added to or removed from the data base.*

This methodology is obviously not without cost. While the additional reasoning required is minimal, caching all of the set size information and the associated justifications does require some storage. It is, however, often much less than the storage overhead of caching base-level facts and their justifications. If a problem has two hundred answers, then caching the results demands storing at least two hundred propositions and two hundred justifications (many more if intermediate results are derived and cached). In contrast, only one cardinality assertion and its justification need be cached for each subproblem encountered.

3.2.3 Invariance

As a practical matter, it is often possible to cut down the overhead of maintaining information about set sizes by specifying *invariance* for static quantities. In the case of the intelligent agent the number of tape drives usually doesn't change. Thus we could specify:

INVARIANT('CARDINALITY(TAPEDRIVES)=ν') (3-13)

If a cached statement is *invariant* (will not change over time) then there is no need to keep its justification around. Thus if a problem solver dynamically calculates the size of a set, but finds that there is an invariance statement (like those above) then the system can cache the statement and use it without storing elaborate justifications.

It is also possible to make more general statements about invariance. For example, the elaborate justification mechanism could be completely defeated by asserting:

INVARIANT(ρ) (3-14)

Probably more useful would be to specify interesting classes of expressions that are invariant. For example we could state that the number of hardware peripherals is invariant:

SUBSUMES(PERIPHERAL,ρ) ⟹ INVARIANT('CARDINALITY({x:ρ(x)})=ν'). (3-15)

Given this, it would only be necessary to specify that something was a peripheral (e.g.: SUBSUMES(PERIPHERAL,PRINTER)) in order to get the property of invariance.

The use of invariance is not limited to the techniques of this paper. Stating invariance can be advantageous whenever a system is dealing with a changing world. Under these circumstances much of the overhead of *Truth Maintenance* (or whatever techniques are used to deal with a changing world) can be eliminated. It has been mentioned here because it is often applicable to statements about set size, and because the justifications and maintenance for these statements are quite messy. Judicious use of these shortcuts can help to keep down both the computational and storage overhead involved in this scheme. The more elaborate techniques of section 3.2.2 will then only come into play for those cases in which they are truly needed.

4. Interaction with other Meta-level Reasoning

When we try to pick out anything by itself we find it hitched to everything else in the universe.

"My First Summer in the Sierra"
-- John Muir

It would be nice if all were as rosy as we have made it out to be in section 3. Unfortunately, there are some very subtle interdependencies that can arise between statements about the number of solutions, and independent meta-level reasoning used to prune search spaces. We will first give a somewhat sketchy description of the difficulty and then illustrate the problem with an example.

Suppose that a problem solver is attempting to find the solutions for a problem P. In so doing, it generates a subproblem Q, which generates an additional subproblem R. Further suppose that, due to independent meta-level reasoning (the details are not relevant) we decide that the sub-sub-problem R cannot possibly contribute any new solutions to the overall problem of finding the solutions to P. As a result the sub-sub-problem R is *pruned* and the problem solving proceeds.

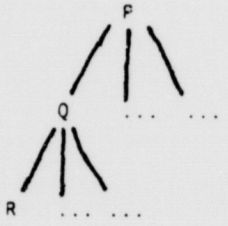

Figure 4-1: Simple Subproblem Tree

Consider what would have happened if we were keeping track of, and caching, the number of solutions to each of these subproblems, (as in sections 3.2.1 and 3.2.2). Suppose that there are three solutions to the problem R, five solutions to the problem Q, and ten solutions to the original problem P. The number of solutions calculated (and cached) for the problem P will still be correct, because we have not eliminated any unique solutions by our pruning. But consider what number will be calculated for the subproblem Q. If all three of the solutions to R contribute to unique solutions for Q, then only two of the solutions to the problem Q will be found, rather than all five. This number is wrong, and if cached, will cause difficulties when the problem Q is encountered again, either in vacuo, or in the context of some other problem.

As a specific example consider two simple facts about familial relationships:

```
CHILD(x,y) & MALE(x) ⇔ SON(x,y)                    (4-1)
CHILD(x,y) & FEMALE(x) ⇔ DAUGHTER(x,y).
```

together with the simple data base:

```
SON(STACEY,MARTHA)                                    (4-2)
SON(BROOKS,MARTHA)
DAUGHTER(JAN,MARTHA)
```

Suppose that the query

find all x *such that* SON(x,MARTHA)

is posed to the system. After discovering the two answers immediately in the data base (BROOKS and STACEY), the subproblem

```
CHILD(x,MARTHA) & MALE(x)                             (4-3)
```

is generated, which generates the subproblem

```
CHILD(x,MARTHA).                                      (4-4)
```

Backward reasoning on the second of the axioms produces one answer (JAN) for this subproblem. The first axiom is also applicable to this subproblem, and would produce two additional answers to the subproblem. But because of the fact that the primary objective is to find Martha's sons, and the first axiom has just been applied in one direction, there is absolutely no point in turning around and applying it in the opposite direction. It won't lead to any new solutions. As a result, that line of inference can be discarded, and only one solution is produced for the subproblem of finding Martha's children.

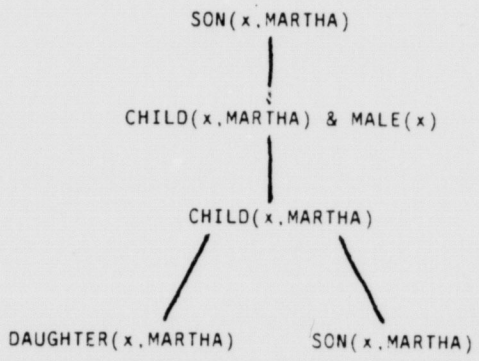

Figure 4-2: Subproblem Tree for Kinship Problem

Ordinarily, this would cause no harm. We still get all of the answers to the original query, and if any intermediate results are cached they will still be correct, but perhaps incomplete. Consider, however, what happens if we blindly apply the methods of section 3.2.2. We would end up caching the following two meta-level facts:

```
CARDINALITY(MARTHAS-SONS) = 2                         (4-5)
CARDINALITY(MARTHAS-CHILDREN) = 1.
```

The first is correct, but the second is wrong. Martha has three children, not one.

Why did this happen? Because a portion of the subproblem was pruned due to consideration of something outside the context of the subproblem, namely, the overall problem being solved.

A solution to this difficulty is to introduce an additional argument in the NUMBEROF relation. If any contextual information is used to "help" solve this subproblem then it must be recorded in the additional argument. This contextual information is, in effect, represented by the *justification* of the meta-level conclusion that "this portion of the subproblem can be eliminated". The details of this justification are dependent upon the specific structure of the problem solver involved (a meta-level decision to prune must involve some reasoning about the problem solving process), as well as the particular technique of meta-level pruning. As a practical matter, it is possible to simply use a T for the context argument in those cases where *external pruning* has taken place. This would mean that these cached meta-level statements (ones with T's) are not useful for estimating the number of solutions when the subproblem is encountered again. Thus, their only purpose is to record the dependency tree for use in truth maintenance. This is probably not a large loss, since, even if full context arguments were kept, it is unlikely that a context argument would match up with any other except that of the original problem, whose results would be cached anyway.

Finally, two things should be noted. First, the cached information for the overall problem will be correct in any case. Second, the extra argument to NUMBEROF statements will always be null if *external pruning* is entirely absent during the course of solving a subproblem. In this case, the proposition about the number of solutions would be the same as if the subproblem had been solved in vacuo.

5. Results

The methods described in this paper have been implemented in experimental versions of the MRS system [10]. To make use of the techniques described, the most stringent requirement is a representation system that permits the expression of the necessary knowledge about set definitions and set sizes. Given this, it is possible to "shoehorn" these methods into most existing problem solvers. While we chose to represent meta-level propositions (e.g. NUMBEROF statements and justifications) explicitly in this paper it is not necessary to do so in order to make use of the techniques described.

Our initial experiences with these methods has been quite positive. For Kunz' model of renal physiology [13], knowledge about the sizes of important sets was provided along with several statements of invariance. While invariant set sizes were cached, the more elaborate storing of justifications outlined in section 3.2.2 was not complete at the time, and no other caching was done. Typical queries of the model usually require on the order of fifteen minutes of CPU time on a DEC 20/60. The addition of knowledge about set sizes along with a few invariance statements resulted in speed increases from two to ten times, depending on the particular query. In spite of the success, the effort was limited by the inability to express certain complex invariance statements to the MRS system available at the time. These deficiencies have since been remedied, making additional speedup possible using only these elementary techniques.

A second application of the techniques presented here arose in the development of a simple income tax consultant [9]. During the consultation, the system needs to make inferences about familial relationships. Because of the many different familial relationships involved (e.g. Mother, Father, Aunt, Uncle, Sister, Brother, Daughter, Son, Sibling, Child, Parent), the search space for even the simplest queries was extremely large. For example, a simple query to determine all of the siblings of a given person generated hundreds of subgoals (fifteen pages of hardcopy trace) and took nearly fifteen minutes of real time to produce the answers (Circular reasoning was pruned as in the example of section 4). The addition of a priori knowledge about the number of parents a person has, and knowledge that MOTHER and FATHER are function symbols, reduced the number of subgoals by a factor of three, and reduced the run time by a similar factor. Use of the full techniques described in sections 3.2.2 and 4 resulted in an additional factor of two reduction when running the problem for the first time. The reason for this speedup is that the same subgoals were often generated several times by different branches of the search tree. The first occurrence would cause the answers and meta-knowledge to be cached. Subsequent occurrences of the subproblem could then make use of the cached knowledge about set size.

6. Discussion

6.1 The Closed World Assumption

Observant readers will note that we have relied on a closed-world assumption throughout this paper. We have tacitly assumed that a problem solver is capable of producing every solution to a problem, and therefore, that the theoretical number of solutions to a problem is the same as the number that the system can produce.

Although convenient, this assumption is not a necessary one. The theoretical number of solutions is an upper bound on the actual number that a problem solver can produce (assuming sound logic and a correct database) and remains useful in any case where the system can produce all of the answers to a problem. Cached information, however, is information about the actual number of answers a problem solver can produce. To distinguish these two concepts, it is sufficient to use a different relation name, say ACTUALNUMBEROF, supply the fact that

$$\text{ACTUALNUMBEROF}(v,p) \leq \text{NUMBEROF}(v,p) \qquad (6\text{-}1)$$

and change our problem solver to use the actual number for cutting off inference.

In systems where the closed world assumption is not valid there is additional advantage to having both kinds of information available. Information about the actual number of solutions permits inference to be stopped, while theoretical information allows a system to warn the user when it cannot produce all of the answers to a problem. As a result of implicit closed-world assumptions, expert systems often generate ludicrous answers to questions outside their area of expertise. Having both kinds of information available allows a problem solver to determine when it can and cannot solve problems, which would eliminate such errant behavior.

6.2 Adding Other Meta-Level Reasoning

Recognizing when all the solutions to a problem have been found is only one of the many possible methods for pruning a search space. Other methods include: restricting facts to be used only in forward or backward inference, eliminating repeated subgoals in an inference, and avoiding "undoing" an inference that has just been done. Aside from the interaction mentioned in section 4, these methods are all essentially independent. There is nothing to prevent them from being used in conjunction.

Meta-level ordering of search spaces actually has a synergistic effect with the techniques presented here. Ordinarily, when a system is asked to find all of the solutions to a problem, ordering of the search space is of no benefit. The entire space must be searched anyway. However, when the number of solutions is known,

ordering of the search space becomes valuable. If all of the answers can be collected rapidly (by searching promising branches of the space first), more of the space will be pruned by a recognition that all of the solutions have been found. Conversely, the utility of knowing the number of solutions depends on discovering all of the solutions before the space is exhausted. For problems in which all the solutions are desired *ordering is of no use without knowing the number of solutions, just as knowing the number of solutions is of no use under the worst possible ordering.*

6.3 Perspective

In this paper a methodology has been presented for shutting off reasoning when all the answers to a problem have been found. To make use of this, knowledge about the sizes of relevant sets must be provided, or derived by the problem solver. It is quite possible for a system to gather such knowledge dynamically and keep it up to date, although the mechanics of truth maintenance are somewhat complex for this case.

This research is a small portion of a much larger effort to demonstrate that meta-level reasoning is an essential component in building intelligent artifacts, as well as a practical methodology for construction of expert systems. Meta-level reasoning is not a single technique that can be knocked off and buried. Rather, it is an entire paradigm. There are, at the very least, hundreds of meta-level problems like the one we have described here. Each one has its own special set of concepts (vocabulary) and governing laws which must be discovered and formalized for use in intelligent systems.

Many authors have argued for the use of meta-level reasoning. More recently several authors have explored general frameworks in which systematic meta-level reasoning is possible [6, 8, 11, 15, 18]. Like that of this paper, there have also been a few attempts to codify the necessary meta-knowledge for solving specific meta-level problems. Among the most notable efforts are those of Bundy [2], Clancey [3], Davis [4], and Wilensky [19]. We regard these efforts (as well as our own) as mere "drops in a bucket". Enormous opportunity remains for significant research into any one of the myriad of outstanding meta-level reasoning problems.

Acknowledgements

Thanks to John Kunz for an interesting and motivating application. Also thanks to Pat Hayes for his reinforcement and excitement. Jan Clayton and Lew Creary provided useful comments on drafts of this paper.

This work was supported by ONR contract N00014-81-K-0004.

References

[1] Barnett, J. A.
How Much is Control Knowledge Worth? A Primative Example.
Working Paper, USC/ISI, May, 1983.

[2] Bundy, A., Byrd, L., Luger, G., Melish, C., Palmer, M.
Solving Mechanics Problems Using Meta-level Inference.
In *Proceedings of the Sixth IJCAI*, pages 1017-1027. International Joint Conference on Artificial Intelligence, August, 1979.

[3] Clancey, W. J., Letsinger, R.
NEOMYCIN: Reconfiguring a Rule-based Expert System for Application to Teaching.
Heuristic Programming Project Memo HPP-81-2, Stanford University, February, 1981.

[4] Davis, R.
Meta-Rules: Reasoning about Control.
AI 15:179-222, 1980.

[5] Doyle, J.
A Truth Maintenance System.
AI 12:231-272, 1979.

[6] Doyle, J.
A Model for Deliberation, Action, and Introspection.
Artificial Intelligence Laboratory Memo AI-TR-581, Massachusets Institute of Technology, May, 1980.

[7] Feigenbaum, E. A., Genesereth, M., Kaplan, S. J., Mostow, D. J.
Intelligent Agents.
1980.
Proposal to the Defence Advanced Research Projects Agency.

[8] Genesereth, M. R., Smith, D. E.
Meta-Level Architecture.
Heuristic Programming Project Memo HPP-81-6, Stanford University, December, 1982.

[9] Genesereth, M. R.
The MRS Casebook.
Heuristic Programming Project Memo HPP-83-26, Stanford University, June, 1983.

[10] Genesereth, M. R.
An Overview of MRS.
Heuristic Programming Project Memo HPP-83-?, Stanford University, June, 1983.

[11] Hayes, P. J.
Computation and Deduction.
In *Proceedings of the Symposium on Mathematical Foundations of Computer Science*, pages 105-117. Czechoslovakian Acadamy of Sciences, 1973.

[12] Konolige, K.
A First Order Formalization of Knowledge and Action for a Multiagent Planner.
Artificial Intelligence Center Technical Note 232, SRI International, December, 1980.

[13] Kunz, J.
Use of Artificial Intelligence and Simple Mathematics to Analyze a Physiological Model.
PhD thesis, Stanford University, 1984.
In preparation.

[14] McCarthy, J.
First Order Theories of Individual Concepts and Propositions.
In Hayes, J. E., Michie, D., Mikulich, L. I. (editors), *Machine Intelligence 9: Machine Expertise and the Human Interface*, pages 120-147. Ellis Horwood, Chichester, 1979.

[15] Smith, B.
Reflection and Semantics in a Procedural Language.
Artificial Intelligence Laboratory Memo AI-TR-272, Massachusets Institute of Technology, January, 1982.

[16] Smith, D. E., Genesereth, M. R.
Ordering Conjuncts in Problem Solving.
Heuristic Programming Project Memo HPP-82-9, Stanford University, August, 1983.

[17] Smith, D. E.
Controlling Inference.
PhD thesis, Stanford University, 1984.
In preparation.

[18] Weyhrauch, R. W.
Prolegomena to a Theory of Mechanized Formal Reasoning.
AI 13(1):133-170, 1980.

[19] Wilensky, R.
Meta-Planning: Representing and Using Knowledge About Planning in Problem Solving and Natural Language Understanding.
Cognitive Science 5(3):197-234, 1981.