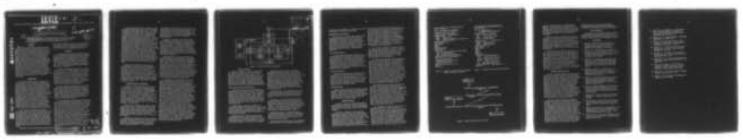
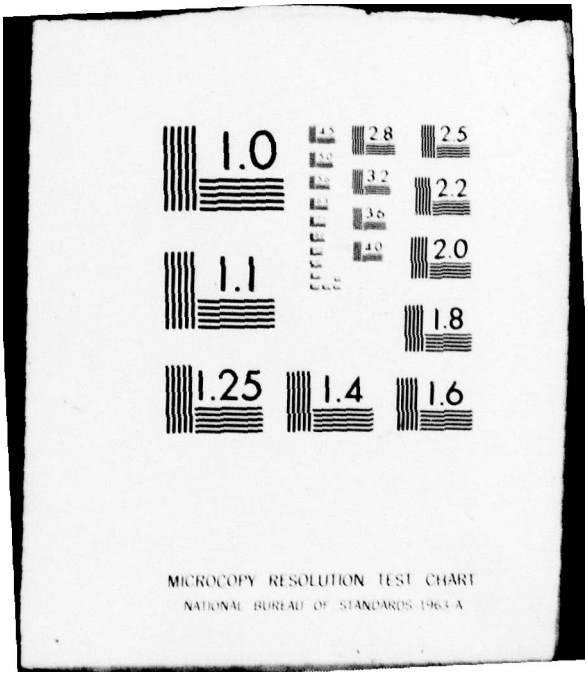
AD-A072 091 SYSTEM DEVELOPMENT CORP SANTA MONICA CALIF F/G 5/2
DEDUCTIVE METHODS FOR LARGE DATA BASES,(U)
AUG 77 C KELLOGG, P KLAHR, L TRAVIS N00014-76-C-0885
UNCLASSIFIED NL

| OF |
AD
A072091



END
DATE
FILMED
9-79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

Fifth International Joint Conference on Artificial Intelligence, MIT, Aug. 22-25, 1977

Contract N00014-76-C-0885

DEDUCTIVE METHODS FOR LARGE DATA BASES

Charles Kellogg, System Development Corporation, Santa Monica, Calif.
Philip Klahr, System Development Corporation, Santa Monica, Calif.
Larry Travis, University of Wisconsin, Madison, Wisconsin

ABSTRACT

The design and prototype implementation of a deductive processor for efficient extraction of implicit information from explicit data stored within a relational data-base system is described. General statements (premises or inference rules) as well as queries are expressed in a canonical form as implications. From user queries, the system constructs skeletal derivations (proof plans) through the use of a predicate connection graph, a pre-computed net structure representing possible deductive interactions among the general statements. The system incorporates techniques for rapid selection of small sets of relevant premises (by proof planning); development and elaboration of proof plans; proof plan verification; use of proof plans as a basis for determining data-base access strategies; and instantiation of plans (i.e., turning proof plans into proofs) with retrieved data-base values. Examples of the current capability of the system are illustrated.

INTRODUCTION

The deductive processor (DP) described in this paper has been designed to interface with existing and emerging relational data management systems (RDMSs). Given this orientation, we have made a sharp distinction between specific facts (n-tuples) which reside in an RDMS data base and general statements (rule-based knowledge or premises) that are directly accessible to the DP. Since the number of general statements that may be required for a practical application is likely to be large (perhaps hundreds to thousands of premises), particular attention has been paid to the development of techniques for the rapid selection of relatively small sets of premises relevant to answering a user's specific request. Premise-selection techniques are automatically invoked when deductive support is necessary to respond to a user's request; otherwise, queries "fall through" the DP and directly drive the RDMS.

This "deductive inference by exception" principle suggests that the DP be viewed as an add-on or enhancement to existing data-base searching capabilities[4]. Such an enhancement can result in a major increase in the power of a data management system by providing a means for extracting and deriving implicit information from data bases of explicit facts. Further, as we shall see, the DP can aid a user in evaluating the utility and/or plausibility of an inferentially obtained answer by displaying the evidence on which the answer is based.

We briefly review some of the relevant work in the field of deductive question answering, outline our approach, describe the several components of our prototype DP, and illustrate by means of two examples the current operation of the system.

APPROACH

Previous approaches to adding deductive capabilities to data management have occurred primarily in the development of question-answering systems (Simmons[14,15] reviews many of these). The primary deductive methods that have been used are set-inclusion logic, e.g., CONVERSE[3] and SYNTHEX[11], techniques based on the "resolution" principle[10], e.g., QA3[2] and MRPPS[9]; procedural-oriented deduction, e.g., SHRDLU[18]; and goal-oriented backward chaining, e.g., MYCIN[16].

The primary difference between these systems and our DP is in our use of planning. Our system creates *deduction plans to guide the generation* of full deductions. We believe such planning to be essential for cutting through the massive number of dead ends and irrelevant inferences which have impaired the performance of earlier systems. Planning becomes even more important for systems involving large numbers of premises. Selection of a manageably small set of possibly relevant premises can be based on such planning.

To this end we have designed and implemented a deductive processor that first builds derivation skeletons which represent possible deduction plans. Once such plans are generated, the system will attempt to instantiate and verify the plans (examine substitutions for variables in premises). We have thus separated the premise-selection process from the process of verifying the consistency of variable substitutions.

The generation of derivation (proof) plans is centered around middle-term chaining[5]. This process finds implication chains from assumptions to goals through the premises. Middle-term chaining combines the processes of forward chaining from the assumptions in a query and backward chaining from the goals in a query. (In the case of no query assumptions, middle-term chaining defaults to backward chaining.) As chaining proceeds in the two directions, intersections are performed on the derived sets. When a non-empty intersection occurs, the system has found an implication chain from an assumption to a goal. The resulting chain is passed on to the proof plan generator, which extracts the premises whose occurrences are involved in the chain. Subproblems may result, requiring further deduction or data-base search. The examples presented below will illustrate these processes.

The chaining process does not operate on the premises themselves but on a net structure called the predicate connection graph (PCG). This graph is abstracted from the premises. When a premise is introduced into the system, the implication connections existing among the predicate occurrences in the premise are encoded into the PCG. Further, the deductive interactions (i.e., unifications[10]) between predicate occurrences in the new premise and predicate occurrences in existing premises are pre-computed and encoded into the PCG. The variable substitutions required to effect the unifications are stored elsewhere, for latter use by the proof plan verifier. Thus, the PCG contains information on the implications within premises and the deductive interactions among the premises. During the generation of middle-term chains and proof plans, the system is aware of the existence of unifications among the premises, but it does not need to generate the unifications nor does it need to examine and combine the variable substitutions associated with the interacting unifications. The former is done by a pre-processor, while the latter is done by the verifier after proof planning.

Although some connection graphs used in theorem-proving systems also contain information on the unifications among general assertions (resolution clauses in these systems), they are not used as a planning tool as is the PCG. The PCG most resembles Sickel's clause interconnectivity graph[13] in that both graphs represent the initial deductive search space and are not changed in the course of constructing deductions. Other graph procedures[7,12] involve adding nodes to graphs as deductions are formed. More detail on the PCG is given in Klahr[5].

REPRESENTATION OF INFORMATION

The basic representation for general assertions (premises) is the primitive conditional[17]. This form is a normalized first-order predicate-calculus implication statement. The antecedent of the implication contains the assumptions (conditions) of the assertion; the consequent contains the goals of the assertion. Conjunctions, disjunctions, and negations can occur on either side of the implication. Each assumption and goal is a predicate occurrence consisting of a predicate (relation) and its argument terms (i.e., variables, constants, or functions).

The primitive conditional was chosen because general assertions are usually formulated in the form of "if...,then..." implications. Users can easily express and understand general assertions in this form and can easily control and understand proofs involving them. Further, this form facilitates system discovery of deductive implication chains.

Variables and constants occurring in premises and queries may be categorized into specific domain classes. For example, a variable "x" might be specified as being a LABORATORY and the constant "Joe" as being a SCIENTIST. In attempting to match argument strings involving these terms, the system will not allow the substitution of Joe for x because they belong to different domains. The use of such semantic information eliminates certain deductive interactions among the premises and thus reduces the search space of possible deductions [5,8].

Semantic information in the form of user-supplied advice can also be given to the system. Advice most typically involves recommendations on the use of particular premises or predicates in finding deductions. For advised premises, the system will try using them whenever possible in the course of constructing a proof. For advised predicates, the system will try chaining through occurrences of them (in premises). In the case of negative advice, specified premises and predicates are avoided in proofs.

Advice may be given for a particular input query or stored in a permanent advice file which the system accesses for each query. Advice statements are in the form of condition-recommendation rules similar to the meta-rules used in MYCIN[1]. The conditions contain information about predicates, constants, and domain classes that may occur in query assumptions and goals. The conditions are matched against the input query and, if they are satisfied, the associated recommendations about the use of certain premises and predicates are activated. Internally, advice is transformed into premise and predicate alert lists (as well as negative alert lists for negative advice), which are accessed in the chaining and proof-planning processes.

In addition to the information used by the deductive processor, there is also a file of specific facts used by a data management system. This latter system searches for and retrieves specific facts needed to resolve subproblems resulting from premises. For our experiments with the prototype deductive processor, we have written a small LISP relational data-base management system. Facts are stored relationally as n-tuples associated with a predicate (relation) name. When a particular predicate occurrence becomes a subproblem, the system has three alternative methods for resolving it; the decision is based on how the user defined the various predicates known to the system. If a predicate is defined computationally by a procedure, the procedure is executed to determine the predicate's truth value. If a predicate is specified by the user as defined primarily by its data-base values, the unresolved predicate is left for data-base search. Otherwise, an unresolved predicate occurrence is given further deductive support through the premises. (Such predicate classification is currently mutually exclusive but need not be. An alternative control structure could try several methods for resolving each subgoal.) The examples below will show the interface between the deductive processor and the data management system, as well as examples of procedurally defined predicates.

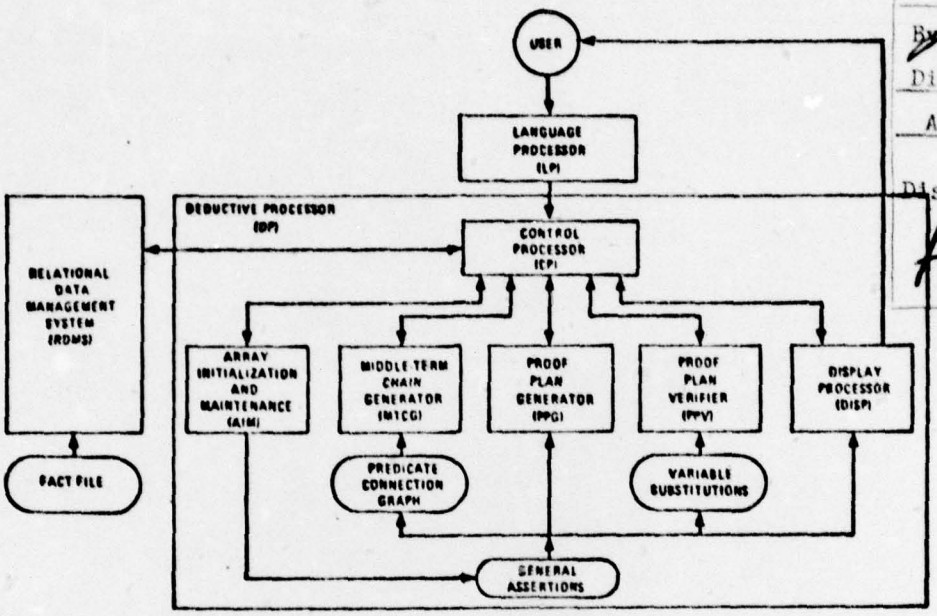


Figure 1. Deductive Processor Components

SYSTEM COMPONENTS

Figure 1 displays the various components of the deductive processor as well as its position in a deductive data management system. The language processor is currently not a part of our initial prototype environment but will be incorporated at a later date. The control processor shown in Figure 1 currently accepts premises and queries in primitive conditional form as well as user advice and commands. It accesses and coordinates the several system components described below.

Array Initialization and Maintenance

Information abstracted from the premises is segmented into seven internal arrays. This segmentation contributes to good system structuring and increases processing efficiency. Each predicate occurrence is assigned a unique integer index. Information about a particular predicate occurrence is obtained from the array containing the kind of information needed by indexing into the array with the integer associated with the occurrence. The seven arrays are:

Premise Array: Each entry represents a premise and contains a list of the occurrences (i.e., occurrence indices) in the premise, the plausibility of the premise, and the premise itself, both symbolic (primitive conditional form) and English, for purposes of display.

Predicate Array: This array contains the relations known to the system. Associated with each relation is its support indicator, i.e., the method used to resolve the relation when it occurs as a subgoal (deduce, search data base, compute).

Predicate Occurrence Array: Each entry represents a predicate occurrence and contains the following information about the occurrence: its predicate name (index into predicate array), the premise in which it occurs (index into premise array), the sign of the occurrence (positive or negative), whether the occurrence is in an antecedent or consequent of a primitive conditional, the main connective governing the occurrence (i.e., conjunction or disjunction), and the numerical position of the occurrence within the premise. The information is compactly stored in a single one-word bit vector.

Arguments Array: The argument strings of the predicate occurrences are stored in this array in a one-to-one correspondence to the positions of the occurrences in the predicate occurrence array.

Links Array: Deductive dependencies within premises are stored in this array. Basically, these dependencies derive from implication connections among predicate occurrences within premises (Klahr[5]). This array is also indexed by occurrence integers. For each occurrence, a list of the occurrences it implies is stored in the entry corresponding to the occurrence's index.

Unifications Array: Each entry contains a list of the unifications (deductive interactions) associated with the given occurrence. The unifications array and the links array comprise the predicate connection graph.

Variable-Substitutions Array: The substitution lists association with unifications are stored in

a one-to-one correspondence to the position of the unifications in the unifications array.

Middle-Term Chain Generator

Each input query is broken down (based on the logical connectives in the query) into sets of assumptions (from query antecedents) and goals (from query consequents). The predicate connection graph is used to find deductive implication chains between assumptions and goals. "Wave fronts" are expanded out of assumptions and out of goals until an intersection is found, at which point the middle-term chain is identified and extracted.

Proof Plan Generator

For each middle-term chain generated, the system extracts the premises whose occurrences are part of the chain. Any subgoals resulting from the premises are set up as requiring deductive support through the premises, data-base search, or procedural computation. Subgoals are added to a proof proposal tree, which contains proof plans as they are being formed and developed. Proof plans having no remaining deduce subgoals are then passed on to the verifier.

Proof Plan Verifier

The variable substitutions required by the unifications in a proof plan are examined for consistency. If there are no clashes, i.e., no variable taking on more than one distinct constant value, then verification is successful. If there are any remaining subgoals requiring data-base support, the data management system is called to search the file of specific facts.

Display Processor

The user has a wide variety of display options available to monitor the operation of the deductive system. In particular, he can examine middle-term chains generated, proof plans formed, subgoals, proof plan verification, data-base search requests, data-base values returned, answers, completed proofs, and premises used in proofs.

COMPUTER EXAMPLES

In Figures 2 and 3 we illustrate examples of the current operation of our initial DP prototype interfaced to a small RDMS. (Both DP and RDMS are written in LISP 1.5 and operate on an IBM 370/158 computer.)

In the first example, we illustrate the generation of short inference and search/compute plans for the question, "What ships are closer to the Kittyhawk's home port than the Kittyhawk is?" The query is first shown in English and then in the primitive conditional symbolic form that our prototype currently recognizes. The query is expressed in terms of a conjunctive goal composed of the predicates CLOSER-THAN and HOME-PORT. Constants (e.g., Kittyhawk) are specified by being enclosed in parentheses, while variables (e.g., x and y) are not. One of the query goals (HOME-PORT) is to be given data-base support, i.e., it has been characterized as defined by data-base values, while the other goal (CLOSER-THAN) is to be deduced. Since the antecedent in the query is empty, middle-term chaining defaults to backward chaining. The system back-chains from CLOSER-THAN through premise 29. The plausibility (similar to certainty factors in MYCIN[16]) of the plan in this case is simply the plausibility of the single premise used. Premise plausibilities range from 1 to 99 and are set by the user.

Two new search requests (in addition to HOME-PORT) result from premise 29, as well as a compute relation containing functional arguments. Computations for the functions and the relation are delayed until values for the variables x and y have been found in the data base (i.e., values which satisfy the search requests).

The system sends the three search requests to the RDMS, which finds two ships, the Forrestal and the Gridley, that are closer to the Kittyhawk's home port (San Diego) than the Kittyhawk is. The system then displays the proof that led to the first answer (the Forrestal). A proof using the other answer would be identical to this one except that Gridley would replace Forrestal in the proof, and the distance between the Gridley and San Diego would replace 310 (the distance between the Forrestal and San Diego). The symbols G2, G3, etc., represent nodes in the proof proposal tree and are used here for reference. G2 and G3 represent the original goals as also shown in the inference plan. G5, G6, and G7 are subgoals that resulted from premise 29, which was used to deduce G2. Thus, these three subgoals are indented below G2.

The middle-term-chaining and proof-planning processes are more evident in the example in Figure 3. The input query contains two assumptions (DAMAGED and DESTINATION) and one goal (TRANSPORT). Taurus and NY are constants; Cargo and x are variables. The query asks the system to find values for x that satisfy the query. The variable x is restricted to range over ships. (This is an example of a domain class specification for a variable. Such domain specifications could also have been used in the previous example.) In the course of developing deductions, the system will not allow values to be substituted for x that belong to domain classes other than ships.

The inference plan shown in Figure 3 has already been verified. To see the planning mechanism more clearly, we will refer to Figure 4. The first middle-term chain generated connects the DESTINATION assumption to the TRANSPORT goal via premise 23. This is shown by the unifications u_1 and u_2 in Figure 4. The predicate occurrences involving the relations AVAILABLE and OFFLOAD become subproblems. The former is to be given data-base support; the latter is deduced by a middle-term chain from the DAMAGED assumption through premises 7 and 15. The chain is shown in Figure 4 by the unifications u_3, u_4, and u_5. The

```
•WHAT SHIPS ARE CLOSER TO THE KITTYHAWK'S HOME PORT
•THAN THE KITTYHAWK IS?

QUERY(((?)IMP(AND(CLOSER-THAN X (KITTYHAWK) Y)
                 (HOME-PORT (KITTYHAWK) Y))))
INFERENCE PLAN:
   DEDUCE   G2 •CLOSER-THAN X KITTYHAWK Y
   SEARCH   G3 •HOME-PORT KITTYHAWK Y
PREMISES USED: (29)      PLAN PLAUSIBILITY:  99
SEARCH/COMPUTE PLAN:
   SEARCH       •SHIPS KITTYHAWK
   SEARCH       •SHIPS X
   SEARCH       •HOME-PORT KITTYHAWK Y
   COMPUTE      •GREATER-THAN (DISTANCE-BETWEEN KITTYHAWK Y) (
                 DISTANCE-BETWEEN X Y)
ENTERING DATA BASE
DATA-BASE SEARCH SUCCESSFUL
••••••••••••••••
ANSWER SUMMARY --
VARIABLES:
(X Y)
ANSWERS:
(FORRESTAL SAN-DIEGO)
(GRIDLEY SAN-DIEGO)
••••••••••••••••
PROOF DISPLAY:
   DEDUCED  G2 •CLOSER-THAN FORRESTAL KITTYHAWK SAN-DIEGO
   FACT     G5 ••SHIPS KITTYHAWK
   FACT     G6 ••SHIPS FORRESTAL
   COMPUTED G7 ••GREATER-THAN 378 310
   FACT     G3 •HOME-PORT KITTYHAWK SAN-DIEGO
PREMISES USED: (29)      PROOF PLAUSIBILITY:  99
TYPE PREMISE NUMBER TO DISPLAY, OR 'END':
29
((ALL X79) (ALL X80) (ALL X81)
 (AND (SHIPS X79) (SHIPS X80))
  (GREATER-THAN (DISTANCE-BETWEEN X79 X81)
   (DISTANCE-BETWEEN X80 X81)))
 IMP (CLOSER-THAN X80 X79 X81))
PLAUSIBILITY:  99
TYPE PREMISE NUMBER TO DISPLAY, OR 'END':
END
END DISPLAY
```

Figure 2. Deduction Involving Deduce, Data-Base Search, and Compute Predicates

```
•IF THE TAURUS WERE DAMAGED WHILE DESTINED FOR NEW
•YORK WITH A CARGO, WHAT SHIPS COULD TRANSPORT THE
•CARGO TO NEW YORK?

QUERY(((WHAT (SHIP . X))
       (AND (DAMAGED (TAURUS))
            (DESTINATION (TAURUS) (NY) CARGO))
       (IMP (TRANSPORT X CARGO (NY))))
INFERENCE PLAN:
   DEDUCE   G2 •TRANSPORT SHIP#X X75 NY
   ASSUME      •DESTINATION TAURUS NY X75

   DEDUCE   G3 ••OFFLOAD TAURUS X76 X72
   ASSUME      ••DAMAGED TAURUS
   MID-TERM    ••RETURNS TAURUS X72

PREMISES USED: (23 7 15)      PLAN PLAUSIBILITY:  80
SEARCH/COMPUTE PLAN:
   SEARCH       •HOME-PORT TAURUS X72
   SEARCH       •CARRY TAURUS X75
   SEARCH       •AVAILABLE SHIP#X X72
ENTERING DATA BASE
DATA-BASE SEARCH SUCCESSFUL
••••••••••••••••
ANSWER SUMMARY --
VARIABLES:
(X)
ANSWERS:
(PISCES)
(GEMINI)
••••••••••••••••
PROOF DISPLAY:
   DEDUCED  G1 •TRANSPORT PISCES OIL NY
   ASSUME      •DESTINATION TAURUS NY OIL

   DEDUCED  G3 ••OFFLOAD TAURUS OIL FREEPORT
   ASSUME      ••DAMAGED TAURUS
   MID-TERM    ••RETURNS TAURUS FREEPORT

   FACT     G11•••HOME-PORT TAURUS FREEPORT
   FACT     G12•••CARRY TAURUS OIL
   FACT     G4 ••AVAILABLE PISCES FREEPORT
PREMISES USED: (23 7 15)      PROOF PLAUSIBILITY:  80
END DISPLAY
```

Figure 3. Deduction Using Middle-Term Chaining

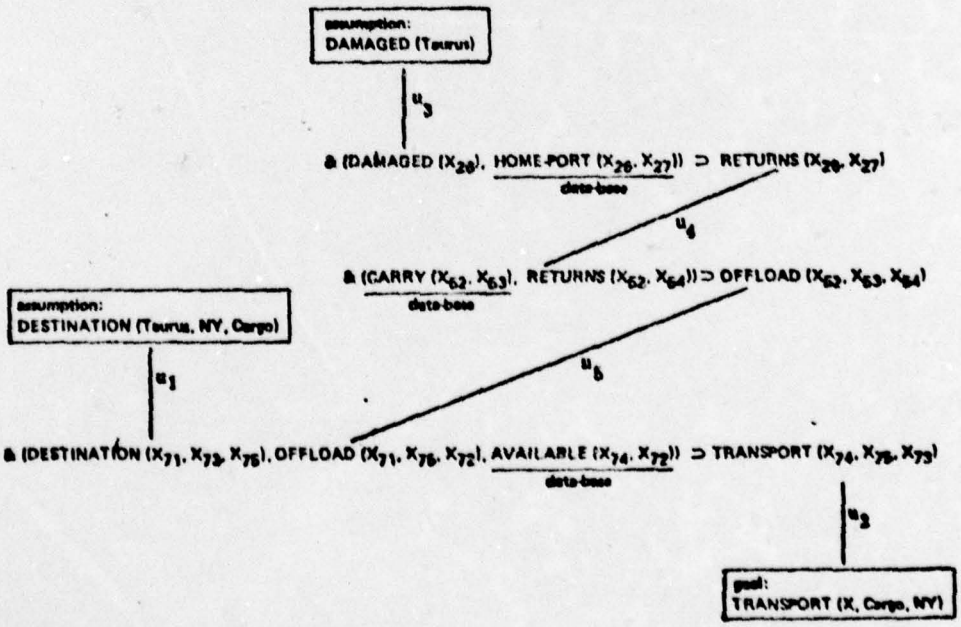


Figure 4. Proof Plan for Query in Figure 3

two new subproblems are to be given data-base support. Thus the plan generated uses three premises and contains three subproblems requiring data-base search. The plausibility of the plan is currently calculated by a fuzzy intersection (the minimum of the plausibilities of the premises involved[19]).

The plan is then verified with variable substitutions inserted in the plan and in the search requests (Figure 3). Note the variable constraints in the search requests. The variable x_{72} represents the home port of Taurus; values found for this variable must be the same as those found for x_{72} in the AVAILABLE search request. The proof display is given for the first answer found (the Pisces).

In Figure 4 we note that the unifications u_4 and u_5 were computed when these premises were first entered into the system and stored in the PCG. Also stored in the PCG were the implication connections within the premises, e.g., between DAMAGED and RETURNS, between RETURNS and OFFLOAD, and between DESTINATION and TRANSPORT. The unifications u_1, u_3, and u_2 were computed after query input (because they involve predicate occurrences in the query) and serve to locate possible middle-term-chain end points. Once these end points were identified, only the PCG was used for middle-term chaining.

SUMMARY AND FUTURE PLANS

We have described a deductive system specifically designed to provide inferential capability for a data management system. From a set of general assertions, the system generates skeletal derivations or proof plans in response to given input queries. These plans are then used to trigger data-base search requests for the specific facts needed to instantiate and thus complete proof plans, turning them into proofs and answers. General information is thus used to guide and direct the proof-planning process and to identify subproblems that may be resolved by data-base search or by computation. (Or subproblems may be left open in the display of incomplete proof plans to the user, thus identifying information which cannot be found within the system but which the user may be able to supply from without.)

We are currently expanding the prototype along several different dimensions in line with our goal of eventually incorporating the deductive processor into an operational data management system and language processor environment. A number of improvements in man-machine interaction and user displays are being made in order to allow users to have more direct and flexible control of the proof-plan-generation and data-base-search processes. Additional semantic constraints on the generation of plans will be introduced through the use of a semantic net to further restrict the range of variables, as well as through extensions to the existing semantic-advice condition-recommendation formalism. Work in these two critical areas of improved user and semantic control of deductive processes is being supplemented by additional investigations into the encoding and integration of incomplete and plausible knowledge.

ACKNOWLEDGEMENTS

The research reported here has been supported by the Advanced Research Projects Agency of the Department of Defense and is monitored by the Office of Naval Research under Contract N00014-76-C-0885.

REFERENCES

1. Davis, R., Buchanan, B., and Shortliffe, E. Production rules as a representation for a knowledge-based consultation program. Artificial Intelligence, 8, 1977, 15-45.

2. Green, C. C. Theorem proving by resolution as a basis for question-answering systems. In Machine Intelligence 4, Meltzer, B. and Michie, D. (Eds.), Edinburgh University Press, Edinburgh, 1969, 183-205.

3. Kellogg, C. H., Burger, J., Diller, T., and Fogt, K. The CONVERSE natural language data management system: current status and plans. Proceedings of the Symposium on Information Storage and Retrieval, ACM, New York, 1971, 33-46.

4. Kellogg, C., Klahr, P., and Travis, L. A deductive capability for data management. In Systems for Large Data Bases, Lockemann, P. C. and Neuhold, E. J. (Eds.), North Holland, Amsterdam, 1976, 181-196.

5. Klahr, P. The Deductive Pathfinder: creating derivation plans for inferential question-answering. Ph.D. Dissertation, Computer Science Department, University of Wisconsin, Madison, December 1975.

6. Klahr, P. Planning techniques for rule selection in deductive question-answering. In Pattern-Directed Inference Systems, Waterman, D. and Hayes-Roth, F. (Eds.), Academic Press, New York, 1977.

7. Kowalski, R. A proof procedure using connection graphs. Journal of the ACM, 22, 1975, 572-595.

8. McSkimin, J. R. The use of semantic information in deductive question-answering systems. TR-465, University of Maryland, College Park, 1976.

9. Minker, J., Fishman, D. H., and McSkimin, J. R. The Q* Algorithm--a search strategy for a deductive question-answering system. Artificial Intelligence, 4, 1973, 225-243.

10. Robinson, J. A. A machine-oriented logic based on the resolution principle. Journal of the ACM, 12, 1965, 23-41.

11. Schwarcz, R. M., Burger, J. F., and Simmons, R. F. A deductive question-answerer for natural language inference, Communications of the ACM, 13, 1970, 167-183.

12. Shostak, R. F. Refutation Graphs. Artificial Intelligence, 7, 1976, 51-64.

13. Sickel, S. A search technique for clause interconnectivity graphs. IEEE Transaction on Computers, C-25, 1976, 823-835.

14. Simmons, R. F. Answering English questions by computer: a survey. Communications of the ACM, 8, 1965, 53-69.

15. Simmons, R. F. Natural language question-answering systems: 1969. Communications of the ACM, 13, 1970, 15-30.

16. Shortliffe, E. H. Computer-Based Medical Consultations: MYCIN. American Elsevier, New York, 1976

17. Travis, L., Kellogg, C., and Klahr, P., Inferential question-answering: extending CONVERSE. SP-3679, System Development Corporation, Santa Monica, Calif., 1973.

18. Winograd, T. Understanding Natural Language. Academic Press, New York, 1972.

19. Zadeh, L. A. Fuzzy sets. Information and Control, 8, 1965, 338-353.