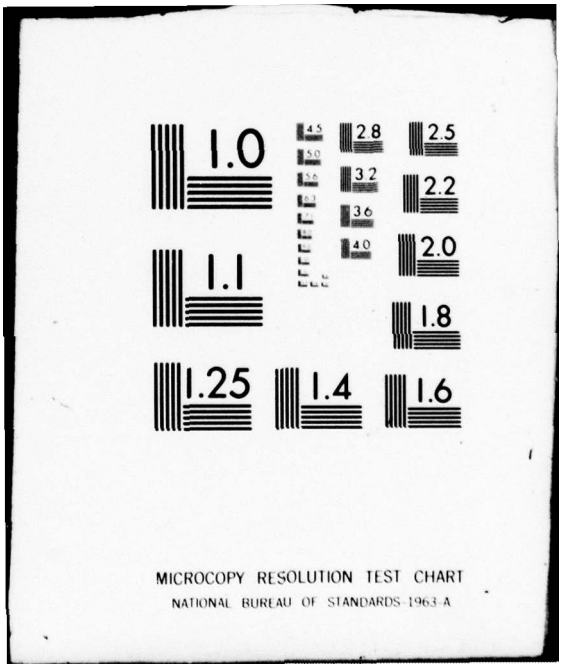
AD-A072 092 SYSTEM DEVELOPMENT CORP SANTA MONICA CALIF F/G 5/1
PLANNING TECHNIQUES FOR RULE SELECTION IN DEDUCTIVE QUESTION-AN--ETC(U)
MAY 77 P KLAHR N00014-76-C-0885
UNCLASSIFIED NL

| OF |
AD A072092

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
4.5
5.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD A072092

code 437
NR. 049-400

LEVEL II

(6) Planning Techniques for Rule Selection in Deductive Question-Answering

(10) Philip Klahr

System Development Corporation
2500 Colorado Avenue
Santa Monica, California 90406

(15) Contract N00014-76-C-0885
Date (11) May 1977

(12) 27 p.

IN

Pattern-Directed Inference Systems
Waterman, D. & Hayes-Roth, F. (Eds.)
Academic Press, NY, 1978

DDC FILE COPY

DDC
AUG 1 1979
A

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

339 900

P. Klahr

ABSTRACT

Deductive planning techniques are described for rule selection in question-answering systems. Rules typically represent inferential knowledge applicable to a given domain of discourse. Given a question-answering system containing a large number of such rules, a crucial problem arises in selecting those rules that are relevant and needed to answer particular input queries. A planning process has been implemented to find chains of rules that deductively infer the desired conclusions. The processes of forward chaining from assumptions and backward chaining from goals are combined to form middle-term chains which provide the basis of the planning mechanism. The applicability and generality of these techniques for use in other rule-based systems is also discussed.

ACKNOWLEDGEMENTS

Much credit is due to my colleagues, Charles Kellogg and Larry Travis, for their many suggestions and insights in the design and development of the deductive system. The research reported here has been supported in part by the Advanced Research Projects Agency of the Department of Defense and monitored by the Office of Naval Research under Contract N0014-76-C-0885.

Accession For
NTIS GRA&I
DDC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Dist. | Avail and/or special
A

1. Introduction

An increasingly important problem in rule-based inference systems is one of selecting rules that are relevant to answer a user's input query or to solve a particular problem. This rule-selection problem becomes crucial in systems containing a large number of such rules. Given a query that needs deductive support, i.e., a query that cannot be answered by direct retrieval from the file of specific facts, the system must find its way among the potentially large set of general statements (rule-based knowledge) and discover a subset that is relevant and sufficient to answer the given query. Without significant guidance and direction, the system may get hopelessly lost in generating irrelevant inferences. New planning techniques have been designed and implemented for selecting relevant rules. Although the techniques have been developed, and will be discussed, within the framework of a question-answering system [11,12], they should be applicable to other systems that use rules as the basis for inferring and deducing new or implicit knowledge.

Early work in question answering used special-purpose deductive mechanisms. The most common was the set-inclusion logic of SIR [19], CONVERSE [10], and SYNTHEX [24]. With the development of the "resolution principle" in mechanized theorem-proving [21], several researchers have incorporated resolution techniques into question-answering systems for purposes of more general-purpose (domain-independent) deduction. Most notable among these was the system developed by Green [8]. But problems with explosive search space, even for small numbers of axioms and theorems, and with canonical forms that are difficult for users to comprehend, have led many researchers to find alternative representations and deductive techniques. Even with the various resolution strategies that have been developed (Chang and Lee [5] discuss most of these), very few current systems use resolution as a basis for question answering (the Maryland system [14] being a notable exception).

As an alternative to resolution, the use of more "natural" deduction techniques have become common in question-answering, problem-solving, and even theorem-proving systems. Such techniques usually involve the use of goal-oriented backward chaining or sub-goal generation as a basis for deduction. Given an input query or goal, rules whose consequents match the goal are invoked, and the antecedents of the rules are set up as subgoals. The process repeats recursively. Such backward chaining techniques are found in several rule-based systems, e.g., in production systems such as MYCIN [6] and RITA [1]; in PLANNER-like language systems [9,4] where rules are in the form of procedures; and in several theorem-proving systems [2,15,20]. Rule selection in these systems ranges from an emphasis on user interaction in choosing rules [3], to having an ordered recommendation list of rules to apply [9], to picking up all applicable rules [6].

The process of backward chaining proceeds in one direction, backward from a goal. In many cases, backward chaining will lead to dead ends, i.e., subgoals which cannot be deduced, found in the data base, or supplied by the user. As backward chaining proceeds, rules are applied if their consequents match the desired goal. Substitutions for variables in the rules occur during the matching process and are passed on to variables in rules selected for further backward chaining. This process thus verifies the consistency of the variable substitutions in the course of selecting and applying rules. But the effort expended in this verification is wasted for those cases where chaining leads to dead ends. The planning mechanism discussed below separates the selection process from the verification process. Verification (consistency of substitutions) is delayed until a set of rules has been selected as being relevant to the particular problem, thus avoiding the verification of many fruitless deductive paths.

The approach to rule selection taken here is one of generating derivation plans, or proof skeletons, representing possible deductions needed to answer queries. The planning process combines backward chaining from goals with forward chaining from

assumptions to form deductive chains through the rules. The generation of these resultant "middle-term" chains is the basic mechanism used for rule selection. In generating middle-term chains, the system is aware of the deductive interactions among the rules but it is not concerned with the verification of variable substitutions within chains or proof plans. Such verification occurs after proof planning. This paper focuses on defining middle-term chaining and showing how it is used in proof planning.

2. Organization and Representation of Knowledge

Within our knowledge base, we will distinguish between facts and rules. Facts are represented in a relational format, i.e., a relation (predicate) followed by its arguments. Examples of facts are GREEK(Socrates), ATTEND(Smith,ACM76), CONDUCTS-RESEARCH-AT(Jones,MIT). Rules are general statements which the system uses to deduce new or implicit knowledge. They are in the form of implications. The left-hand side of a rule represents the set of assumptions or conditions of the rule, while the right-hand side represents the set of goals or actions of the rule. Conjunctions, disjunctions, and negations can occur on either side of the implication. An example rule is &(ATTEND(x,z), ATTEND(y,z)) $\supset$ SCIENTIFIC-CONTACT(x,y), which states that if two scientists (x and y) attend the same conference (z) then scientific contact can occur between them. All variables occurring in rules in this paper will be considered universally quantified (with optional domain class restrictions, e.g., scientist, conference, etc.). For the use of rules containing existentially quantified variables, see Klahr [12].

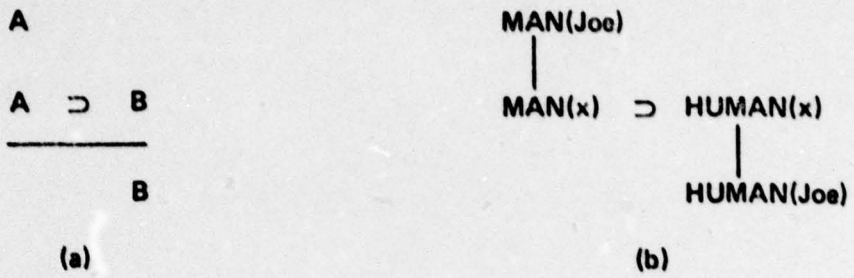
The primary motivation for distinguishing rules and facts is that the deductive processor operates on the rules, while a data management system accesses and retrieves facts. This organization is in sharp contrast to most resolution-based systems, which combine rules (axioms and theorems) and facts into a single file. The distinction between rules and facts being made here is similar to the separation

of rules and data base found in typical production systems [7]. However, the control structure and the interface between rules and data base is more constrained in our system. We are concerned with domains containing large numbers of rules and facts. Our primary focus is on developing techniques for locating relevant rules within our deductive processor. The problem of searching over a large file of facts is also a significant problem, but it is not one we are addressing here. The data management system we have written is satisfactory for use in our initial experimentation, but we are seeking to interface the deductive system with a more sophisticated data management system for retrieval over much larger files of specific facts. We want this interface to be efficient and infrequent and to be used only when facts are needed in a proof. The last example presented in this paper involves such an interface.

3. Deductive Interactions

Before describing middle-term chains and proof planning, we will first show some simple examples of the use of rules in deduction. Figure 1a shows that if we know or assume some proposition A and if we have a rule that states that A implies B, then we can infer B. Figure 1b shows that we can deduce that Joe is a human, given that he is a man and that all men are human. This deduction requires the substitution of "Joe" for the variable x. This substitution is found by a pattern matcher which operates on the argument strings of the relation being considered. We will refer to a successful pattern match as a "unification" (after Robinson [21]). Unifications represent deductive interactions and are shown as vertical lines in Figure 1 and in subsequent figures.

In Figure 1c, two rules are needed to deduce that Joe is a mammal. The two rules deductively interact through a unification between the two occurrences of the predicate HUMAN. Note the consistency of the variable substitutions in the three unifications: The variable x must equal Joe; the variable y must equal x; and y



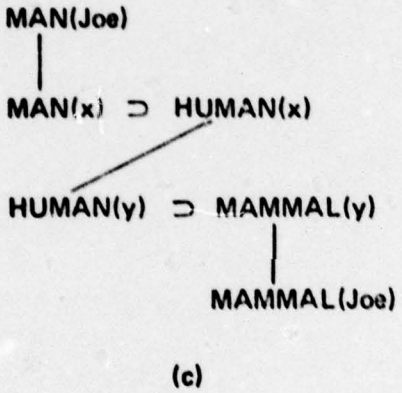


Figure 1. Simple examples of rule-based deduction.

must equal Joe. We will later see how these variable substitutions are combined and used in proof-plan verification. Note also the structural pattern in Figure 1c, a unification followed by an implication, followed by a unification, etc., forming a deductive implication chain between MAN(Joe) and MAMMAL(Joe) through the two given rules.

4. Middle-Term Chains

Consider the example in Figure 2. For purposes of simplicity, only predicate names are shown in the query and in the rules. The argument strings of the predicates have been suppressed. The input query is broken down into the assumption A and the goal D. Suppose the three given rules are known to the system. The subscript attached to each predicate serves to identify the rule number in which the predicate occurs. Suppose that unifications exist between the two occurrences of B and between the two occurrences of C. Suppose, further, that unifications exist between the assumption A and the occurrence of A in rule 1, i.e., A_1, and between the goal D and D_3. Then we have the middle-term chain shown in Figure 2. A_1, which unifies with the assumption, implies B_1, which, in turn, unifies with B_2, which implies C_2, etc. The relations B and C are middle-term predicates, i.e., relations needed to deductively link the assumption to the goal. The predicate occurrences B_1, B_2, C_2, and C_3 are middle-term predicate occurrences. A_1 and D_3, the chain end points, are occurrences of the assumption and the goal, respectively.

The proof plan involving this middle-term chain is also shown in Figure 2. The plan becomes a completed proof if the variable substitutions involved in the unifications are consistent, i.e., no variable takes on two different constants as its value. Plan verification will be shown in later examples.

5. Predicate Connection Graph

In generating middle-term chains, the system uses a net structure called the

Query: $A \supset D$

Assumption: A

Goal: D

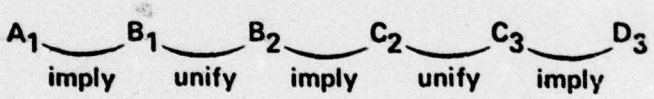
Rules:

(1) $A_1 \supset B_1$

(2) $B_2 \supset C_2$

(3) $C_3 \supset D_3$

Middle-Term Chain:



Proof Plan:

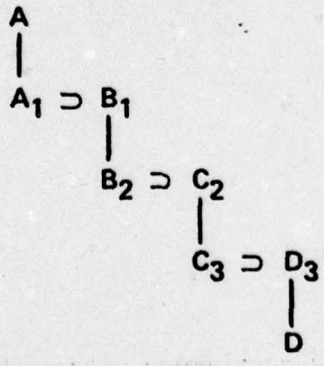


Figure 2. Middle-term chaining.

predicate connection graph. This graph is an abstraction of the rules known to the system. It contains information about the deductive interactions among the rules and the implications within the rules. As rules are entered into the system, unifications between them and the other rules are pre-computed and stored. The existence of unifications is stored in a unifications array while the substitution lists associated with the unifications are stored in a variable-substitutions array. The chain-generation and proof-planning processes use the information about the existence of unifications but need not be concerned with the substitution lists required by the unifications. Once a global plan has been developed, the plan verifier will combine the substitution lists of the unifications in the plan and check for substitution consistency.

The predicate connection graph bears some resemblance to the graphs used in the proof procedures of several theorem-proving systems [13,25,26]. The similarity lies mainly in representation, i.e., using a graph to represent deductive interactions. The primary difference is in the way the graph is used. Here the graph is used as a planning tool to generate possible deductive paths through the rules. The emphasis is on using the graph to locate potentially relevant rules from a large set. The graph does not, by itself, generate proofs, as it does in these other systems.

For the simple example in Figure 2, we note that if the assumption A and the goal D were removed from the proof plan shown, the resulting structure would essentially be identical to the predicate connection graph that would exist for the three given rules. The chain-generation process finds deductive paths through this graph. For larger rule sets, a proof plan would typically be a small subset of the deductive interactions contained in the predicate connection graph.

It is important to summarize the use of the pattern matcher and the use of the substitution lists that result. First, the pattern matcher is invoked when a

new rule is entered into the system. Unifications of the new rule with existing rules are determined and stored. Second, for a given input query, the pattern matcher determines possible chain initialization points, i.e., it finds predicate occurrences within the rules which unify with query assumptions and goals. And third, after proof planning, the substitution lists of the unifications in the plan are combined and verified. During proof planning the existence of unifications, as determined by the pattern matcher, is used, but the pattern matcher itself is not invoked.

Thus most of the work done by the pattern matcher occurs in a pre-processor for entering rules into the system and in proof-plan verification, which fills out the details of the proof. This is in contrast to most resolution-based systems, production systems, and PLANNER-like systems which continually verify substitutions in the course of finding a proof or answering a given query. Although the idea of suppressing details is not new (e.g., Newell, Shaw, and Simon [16]), very few systems use different stages of processing to operate on varying levels of detail. A notable exception is the work of Sacerdoti [22,23], whose problem-solving systems show considerable promise in the use of global planning and the suppression of details for later processing.

6. Forward and Backward Chaining

The middle-term chaining process can be visualized as one which generates expanding wave fronts forward from assumptions and backward from goals. The query assumptions are unified with occurrences of the same predicates in the rules. The successfully unified occurrences represent possible chain starting points. Similarly, those occurrences unifying with query goals represent possible chain end points. From then on the system uses the predicate connection graph exclusively to find implication links and unifications in each

direction. In the forward direction, implication links from the unified assumption occurrences are extracted from the predicate connection graph. Then occurrences unifying with the implied occurrences are extracted, etc. The same process occurs backward from the goal occurrences. When an intersection occurs between the two wave fronts, the system has found a middle-term chain. (See Nilsson [17] for a general discussion of graph searching techniques and Pohl [18] for strategies in bi-directional graph searching and intersecting.)

Let us look at an example of how the chain in Figure 2 was found for the given query from a larger set of rules. Suppose that in addition to the three rules shown in Figure 2, the following rules also existed in the system: (4) $A_4 \supset E_4$, (5) $E_5 \supset F_5$, (6) $G_6 \supset H_6$, and (7) $H_7 \supset D_7$. Suppose also that unifications exist between E_4 and E_5, between H_6 and H_7, between A_4 and the query assumption A, and between D_7 and the query goal D. The predicate connection graph for the seven rules is shown in Figure 3. Possible chain starting points are A_1 and A_4. Possible chain end points are D_3 and D_7. Forward chaining from A_1 and A_4 yields B_1 and E_4. Unifications from B_1 and E_4 are B_2 and E_5. Implications from B_2 and E_5 are C_2 and F_5. Backward chaining from D_3 and D_7 yields C_3 and H_7. Unifications from C_3 and H_7 are C_2 and H_6. There is now an intersection point between the two wave fronts at C_2. The resulting chain is A_1-B_1-B_2-C_2-C_3-D_3. In actual operation, forward chaining and backward chaining proceed alternately one step at a time until a chain is found (or until effort limits have been exceeded). The chain found is the only one that exists for this example set of seven rules. Forward chaining to F_5 is a dead end as is backward chaining to H_6 (and further back to G_6).

Note that the chaining processes found in most other systems will verify as they proceed, i.e., substitutions for variables in the argument strings will be found and combined as the system proceeds from A_1 to B_1 to B_2, etc. Similarly they would also verify as they proceed from A_4 to E_4 to E_5 to F_5. Our system

Rules:

(1) $A_1 \supset B_1$

(2) $B_2 \supset C_2$

(3) $C_3 \supset D_3$

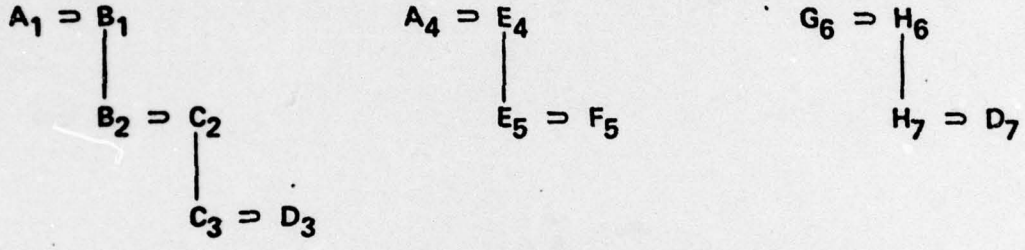
(4) $A_4 \supset E_4$

(5) $E_5 \supset F_5$

(6) $G_6 \supset H_6$

(7) $H_7 \supset D_7$

Predicate Connection Graph:



Middle-Term Chaining:

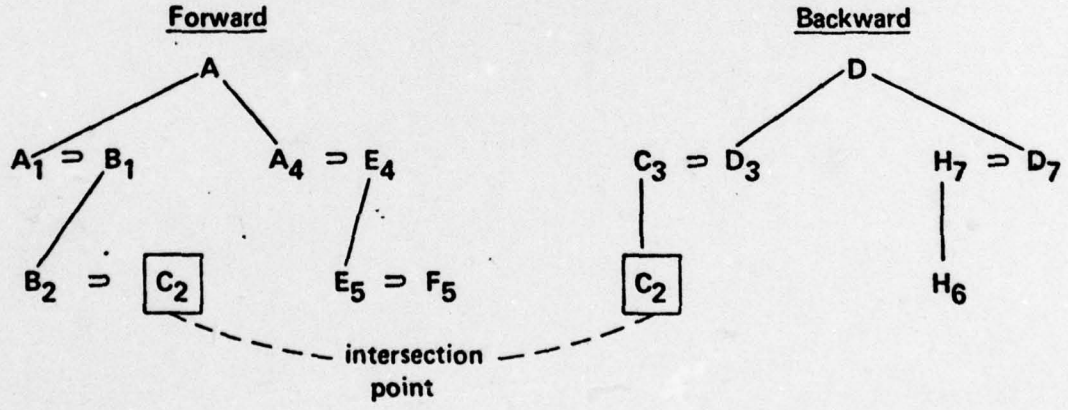


Figure 3. Middle-term chaining through a larger rule set.

verifies only after a proof plan (in this case the single middle-term chain) has been found. The verification of dead ends in this example is avoided.

The process of chain generation is quite fast, owing largely to the representation and storage of information within the predicate connection graph. Each predicate occurrence in the rules is assigned a unique integer, e.g., A_1 is assigned 1, B_1 is 2, B_2 is 3, etc. The set of unifications out of a particular occurrence is given simply by a list of integers, those representing the unifying occurrences. The unifications are stored in an array indexed on the occurrence integers. Thus, to find the unifications existing out of the predicate occurrence whose assigned integer is 2, i.e., B_1, the system simply accesses the second element in the unifications array. A similar process occurs in the storage and retrieval of implication links occurring within rules. These implications are stored in another special-purpose array. Thus, once chain initialization points are found, the system executes its forward and backward chaining by indexing into the two arrays.

The segmentation of information into different arrays has greatly facilitated the organization and processing efficiency of the system. The plan verification processor, for example, uses only the variable-substitutions array to retrieve the substitution lists of the unifications involved in a proof plan. Such segmentation will be of great importance when main memory cannot accommodate a very large set of rules. The various arrays can be swapped in and out of main memory along with the associated processor. In the current version of the deductive system, seven arrays are used. Implementation details are given in Klahr [12].

Even with such efficient storage and retrieval of deductive and implication information, the expanding wave fronts can become quite large, especially if there are a large number of deductive interactions among the rules. Three primary methods are used in reducing and ordering the implications and unifications picked up at any point in middle-term chaining. These include the use of semantic information, user-supplied advice, and plausibility measures.

The pattern matcher finds unifications between predicate occurrences by finding substitution lists that make the argument strings of the occurrences identical. While matching two argument strings, the pattern matcher also checks the domain classes of the variables and constants being matched. When rules or queries are entered into the system, the user can specify domain classes for any variables or constants involved. For example, he could specify the variable x as being a scientist, the variable y as being a politician, and the constant "Einstein" as being a scientist. Then the pattern matcher would allow the substitution of Einstein for x, but would not allow the substitution of Einstein for y nor the substitution of x for y, since they belong to different domains. Domain-class specifications can also include conjunctions, e.g., scientist and teacher; disjunctions, e.g., scientist or politician; and negations, e.g., not scientist. The use of such semantic domain types can greatly reduce the number of deductive interactions that exist among the rules as well as reduce the number of chain-initialization points for a given query.

We are currently expanding the use of domain types to include domain subsets and supersets. This would allow successful matches to occur between elements belonging to domains in which one domain is a subset of the other, e.g., between a variable that is a scientist and one that is a human.

The system also allows the user to supply advice to the system. Advice can be entered for a particular query or it can be stored in a more permanent advice file which the system accesses for each goal. The user can specify that particular predicates and rules be used in deductive chaining. The advice is transformed into predicate and rule alert lists (as well as negative alert lists for negative advice), which serve to order implications and unifications within middle-term chaining. In the case of advised predicates, the system attempts to find chains through occurrences of those predicates. For advised rules, the system tries to chain through them whenever possible. Unifications are ordered at each point in

the chaining process so that unifications to advised rules are given priority. Ordering rules in this way is similar to the use of meta-rules to order rule-selection in [6], although here it occurs in proof planning.

The use of advice gives a user much flexibility. If he can aid the system by specifying particular rules or predicates as being important, the system can take advantage of this. In addition, he can try to find alternative proofs using different advice to see if certain rules are appropriate or to get alternative derivations. In question-answering systems it is often useful to find alternative proofs to give more credence to the conclusion, particularly if rules have varying degrees of plausibility. The system does not necessarily stop with the first completed proof. At the user's request, it will continue to find alternative derivations with or without advice.

The final method used in ordering and selecting elements in middle-term chaining is through the use of plausibility measures. These measures are similar to the certainty factors used in MYCIN [6]. The plausibility of a rule represents the degree of certainty that the antecedents of the rule imply the consequents. Plausibilities can be reset by the user at any time. Unifications are ordered during chaining so that those associated with highly plausible rules are picked up first, although advised rules are given highest priority regardless of plausibility.

7. Generating and Verifying Proof Plans

For each middle-term chain generated, the system extracts the rules containing the chain occurrences and determines if any subproblems result from the rules. In Figure 4, the middle-term chain shown is the same as the one generated in Figure 2. (Once again, argument strings have been suppressed.) However, the rules are more complex. In the first rule, both E_1 and A_1 are needed as conditions for inferring B_1. Thus E_1 is set up as a subproblem. In the second rule, F_2 is not

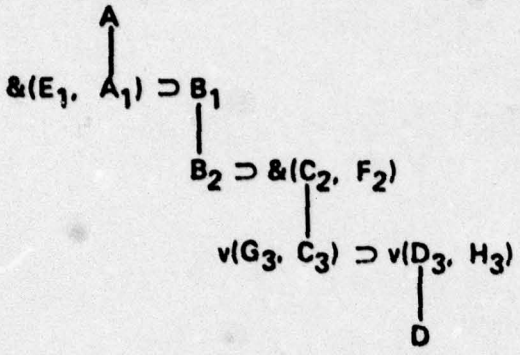


Figure 4. Chaining through more complex rules.

a subproblem since B_2 infers C_2 independently of F_2. Similarly, G_3 is not a subproblem in the third rule. H_3, however, is a subproblem. To infer D_3 by this proof, we must show the negation of H_3 to be true.

Each subproblem is set up for either deductive support through the rules, for data-base support from the file of specific facts, or for computation. The latter occurs for predicates which are defined by user-supplied procedures. When such a predicate occurs as a subproblem, its corresponding procedure is executed to determine its truth value (e.g., predicates such as GREATER-THAN). The user can also specify that certain predicates be given data-base support. These would typically be predicates that are completely defined by the set of facts to which they apply, or predicates about which the user can supply the missing information. In the latter case the system would output conditional answers, i.e., answers which can be concluded if the user can supply the remaining information. All other predicates occurring as subproblems are set up for deductive support to be obtained by recursive calls on the deductive system.

Let us look at more concrete examples. Consider the example shown in Figure 5. The system needs to find a value for y, the place where Socrates lives, given the two assumptions. Suppose the system found the middle-term chain given by the occurrences involved in the unifications u_1, u_2, and u_3. The two rules shown state that if someone is the husband of another then they are married; and married people live in the same place. The resulting subproblem LIVES-IN(x_4,x_5) is resolved directly from an assumption as shown by unification u_4. The resulting structure represents a plan and is not yet a proof, since we have not analyzed the variable substitutions in the plan. This is done by the plan verifier when there are no remaining deductive subproblems (although data-base and compute subproblems may still exist).

The plan verifier extracts the substitutions of the unifications in the plan and creates <u>variable-flow classes</u>. Each class specifies a list of elements, i.e.,

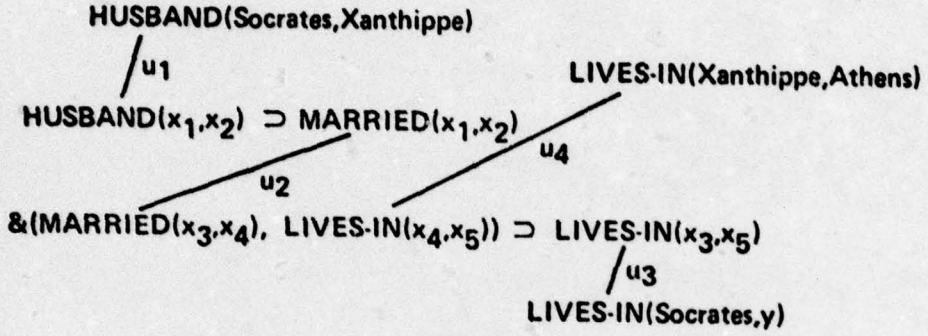
Query: &(HUSBAND(Socrates,Xanthippe), LIVES-IN(Xanthippe,Athens))

⊃ LIVES-IN(Socrates,y)

Assumptions: HUSBAND(Socrates,Xanthippe), LIVES-IN(Xanthippe,Athens)

Goal: LIVES-IN(Socrates,y)

Proof Plan:



Variable-Flow Classes: (Socrates, x_1, x_3) (Xanthippe, x_2, x_4)

(Athens, x_5, y)

Data-Base Requests: None

Answer: y = Athens

Figure 5. Deduction showing proof plan and variable-flow (verification) classes.

variables and constants, that must be equal for the proof to be logically valid. In the example, x_1 must equal Socrates from unification u_1. From u_2, it must also equal x_3, which, in turn, must equal Socrates from u_3. The variable-flow class (Socrates, x_1, x_3) is thus formed. Following another variable flow, we see that x_2 equals Xanthippe from u_1, and equals x_4 from u_2. The variable x_4, in turn, equals Xanthippe from u_4. The resulting variable-flow class is (Xanthippe, x_2, x_4). Finally, if we follow the flow of y, we see that from u_3, y equals x_5 which, in turn, equals Athens from u_4, forming the class (Athens, x_5, y). Note that in each class there is at most one distinct constant. If any variable-flow class contains two different constants, the proof plan fails to verify. In this example, there are no substitution inconsistencies, and the plan successfully verifies.

Since the proof plan contains no remaining subproblems needing evaluation or data-base search, the plan is a complete proof. The system then outputs an answer. If variables were specified in the input query, values for these variables would be displayed. If no variables exist, the system would respond affirmatively if it found a successful derivation. In the example, the variable y occurs in the query. The system locates the variable-flow class in which y occurs. Since the constant "Athens" is in the same class, it becomes the value of y and the answer to the query.

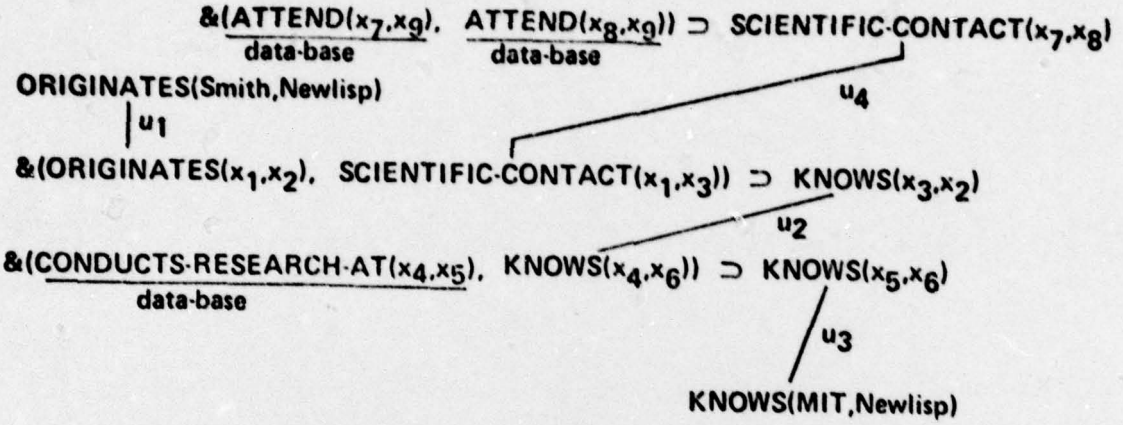
Consider the example in Figure 6. The query asks whether MIT knows about Newlisp which was originated by Smith. The three rules used are concerned with knowledge transfer among scientists and laboratories: If the originator of a result has scientific (professional) contact with another scientist, the latter will know of the result. Scientific contact between scientists can exist if they attend the same conference. If a scientist knows a certain result and conducts research at a particular laboratory, then the laboratory also knows the result.

Query: ORIGINATES(Smith, Newlisp) ⊃ KNOWS (MIT, Newlisp)

Assumption: ORIGINATES(Smith, Newlisp)

Goal: KNOWS(MIT, Newlisp)

Proof Plan:



Variable-Flow Classes: (Smith, x_1, x_7) (Newlisp, x_2, x_6)
(MIT, x_5) (x_3, x_4, x_8) (x_9)

Data-Base Requests: ATTEND (Smith, x_9)
ATTEND (x_3, x_9)
CONDUCTS-RESEARCH-AT (x_3, MIT)

Figure 6. Deduction requiring data-base facts.

The middle-term chain generated involves the unifications u_1, u_2, and u_3. Two subproblems are formed. The occurrence of CONDUCTS-RESEARCH-AT is to be given data-base support. This was specified previously by the user either because the data-base has considerable knowledge about the relation or because it can be easily determined or found by the user (e.g., knowledge about where researchers work).

The other subproblem involves SCIENTIFIC-CONTACT which is to be given deductive support. The system was unable to find a middle-term chain for this subgoal, so it chained back to the rule shown via unification u_4. (Backward chaining will occur whenever middle-term chains cannot be found between assumptions and goals or when no assumptions exist. In such cases, the middle-term chain generator defaults to using only backward chaining. Techniques are being developed whereby the system can generate appropriate assumptions for goals when no assumptions exist so that middle-term chaining can be attempted. One such technique involves finding facts whose predicates apply to constants which occur in the goal, and using these predicates as assumptions [12].) Two more subproblems result, both requiring data-base support. The resulting proof plan has three subgoals to be resolved by data-base search.

The plan verifier forms the variable-flow classes. No conflicts occur, and the data-base subproblems are set up with respect to the classes. The subproblem ATTEND(x_7,x_9) becomes ATTEND(Smith,x_9), since the constant "Smith" is in the same class as x_7. ATTEND(x_8,x_9) becomes ATTEND(x_3,x_9). In this case, the variable-flow class containing x_8 does not contain a constant. The variable is then replaced by the first member in its class. This is done so that all variables within the same class will be specified as being identical for data-base searching. Thus x_3 is also substituted for x_4 in CONDUCTS-RESEARCH-AT(x_4,x_5), which becomes CONDUCTS-RESEARCH-AT(x_3,MIT), MIT being substituted for x_5. Thus, the deductive system

can conclude that MIT knows about Newlisp if there is a scientist who works at MIT and who also attended a conference which Smith attended.

The data-base search requests are then sent to a data management system (currently a relational data management system written in LISP by the author [12]), which attempts to find values for the open variables which would satisfy the subproblems. For example, if it found that Smith attended ACM76, which Jones also attended, and that Jones works at MIT, then the system would respond "yes" to the original query. If insufficient information about the subgoal relations existed in the data base, the system would output a conditional answer. In a sense, the deductive system can be used without any data base. All data-base search requests would be simply given to the user as remaining subproblems for him to resolve.

We can also use the example in Figure 6 to show how variable and constant domain-class specifications can be used to reduce search space. For example, a user could specify that x_1, x_3 and x_4 are scientists. These variables occur in predicates concerned with scientists originating results, knowing results, working at laboratories, etc. Similarly, x_5 could be specified as being a laboratory, indicating that KNOWS(x_5,x_6) refers to laboratories knowing results. In the query, MIT could be typed as a laboratory. The pattern matcher will check domain-class compatibility when finding deductive interactions. Thus MIT would match x_5, since they are of the same domain type, but would not match x_3. KNOWS(MIT,Newlisp) will thus unify with KNOWS(x_5,x_6) but will not unify with KNOWS(x_3,x_2), thereby reducing the number of potential unifications.

8. Summary and Results

The main focus of the research described here has been on developing techniques to find relevant rules in deductive question answering. However, we feel that these techniques can be used in most rule-based systems, including those using

production rules and those using rules in the form of predicate calculus implications. The processes of forward chaining and backward chaining have been used in previous rule-based systems, but here they are combined to find middle-term chains which form the basis of a planning process designed primarily for rule selection. Middle-term chains are found through the use of a predicate connection graph which contains information about the existence of deductive interactions between rules and the existence of implications within rules. Middle-term chains are expanded into proof plans which are verified when there are no remaining subgoals that require deductive support. Remaining data-base subgoals are sent to a data management system for retrieval from the file of specific facts. Answer extraction is straightforward, given the variable-flow classes produced during plan verification and updated after data-base search (returned data-base values for variables are added to the variable-flow classes).

The primary features of the deductive system may be summarized as follows:

1. Separation of rules and facts; deductive system operates on the rules while a data management system retrieves needed facts.
2. Pre-computation of deductive unifications between new rules and existing rules when rules are added to the system; efficient storage and retrieval of deductive information within a predicate connection graph.
3. Process of middle-term chaining to find deductive paths through the rules. When there are no assumptions or when there are no goals, middle-term chaining defaults to backward chaining or forward chaining, respectively.
4. Separate processing phases for rule selection (planning using middle-term chains) and verification.
5. Use of semantic domain classes to reduce search space.
6. Use of user guidance about which rules or predicates may be appropriate.

The deductive planning system is implemented in LISP and is operational on an IBM 370/158. Our initial experimentation involves the use of about thirty rules containing approximately forty deductive interactions. Results so far have been promising. Queries requiring the use of six or seven rules average about one second of processing time which includes a call on the data management system to resolve about four data-base subgoals. A more comprehensive set of rules is being developed for more extensive testing. One of the problems in generating a large number of rules is that of knowledge acquisition and finding experts in particular knowledge domains to aid in the generation of appropriate and meaningful rules.

We are also concerned with computational comparisons with other deductive systems. But such comparisons are difficult to generate and obtain. Our system involves the generation and then validation of proof plans. This approach is in sharp contrast to most other deductive systems. There is a difference in emphasis. Our system attempts to find relevant rules before those rules are actually used (verified). Thus it would be difficult to compare our system with, for example, resolution techniques which have comparison measures such as number of clauses generated, proof level, etc., for an example set of queries. For a small number of rules, other deductive approaches may be more efficient, since rule selection may be less important. But as the number of rules increases, the problem of rule selection becomes more significant, and we feel certain that some form of planning must be done. We are encouraged by our progress and results, and feel that the planning techniques that we have developed hold considerable promise for rule-based deduction.

References

1. Anderson, R. H. and Gillogly, J. J. Rand intelligent terminal agent (RITA): design philosophy. R-1809-ARPA, Rand Corporation, Santa Monica, 1976.

2. Bledsoe, W. W., Boyer, R. S., and Henneman, W. H. Computer proofs of limit theorems. Artificial Intelligence, 3, 1972, 27-60.

3. Bledsoe, W. W. and Bruell, P. A man-machine theorem-proving system. Artificial Intelligence, 5, 1974, 51-72.

4. Bobrow, D. G. and Raphael, B. New programming languages for artificial intelligence research. ACM Computer Surveys, 6, 1974, 153-174.

5. Chang, C. L. and Lee, R. C. T. Symbolic Logic and Mechanical Theorem Proving. Academic Press, New York, 1973.

6. Davis, R., Buchanan, B., and Shortliffe, E. Production rules as a representation for a knowledge-based consultation program. AIM-266, Stanford Artificial Intelligence Laboratory, Stanford University, 1975.

7. Davis, R. and King, J. An overview of production systems. AIM-271, Stanford Artificial Intelligence Laboratory, Stanford University, 1975.

8. 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.

9. Hewitt, C. Procedural embedding of knowledge in PLANNER. Proceedings of the Second International Joint Conference on Artificial Intelligence, British Computer Society, London, 1971, 167-184.

10. 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.

11. 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.

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

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

14. 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.

15. Nevins, A. J. A human oriented logic for automatic theorem proving. Journal of the ACM, 21, 1974, 606-621.

16. Newell, A., Shaw, J. C., and Simon, H. A. Report on a general problem-solving program for a computer. Proceedings of the International Conference on Information Processing, UNESCO, Paris, 1960, 256-264.

17. Nilsson, N. J. Problem Solving Methods in Artificial Intelligence. McGraw-Hill, New York, 1971.

18. Pohl, I. Bi-directional search. In Machine Intelligence 6, Meltzer, B. and Michie, D. (Eds.), Edinburgh University Press, Edinburgh, 1971, 127-140.

19. Raphael, B. A computer program which "understands". Proceedings of the Fall Joint Computer Conference Vol. 26, Spartan Press, Baltimore, 1964, 577-589.

20. Reiter, R. A semantically guided deductive system for automatic theorem-proving. Advance Papers of the Third International Joint Conference on Artificial Intelligence, Stanford Research Institute, Menlo Park, 1973, 41-46.

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

22. Sacerdoti, E. D. Planning in a hierarchy of abstraction spaces. Artificial Intelligence, 5, 1974, 115-135.

23. Sacerdoti, E. D. The nonlinear nature of plans. Advance Papers of the Fourth International Joint Conference on Artificial Intelligence, 1975, 206-214.

24. 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.

25. Shostak, R. F. Refutation graphs. Artificial Intelligence, 7, 1976, 51-64.

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