1984

A focus of attention algorithm for expert systems

Kevin W. Whiting

Department:

A FOCUS OF ATTENTION ALGORITHM FOR EXPERT SYSTEMS

BY

KEVIN WAYNE WHITING, 1955-

A THESIS

Presented to the Faculty of the Graduate School of the

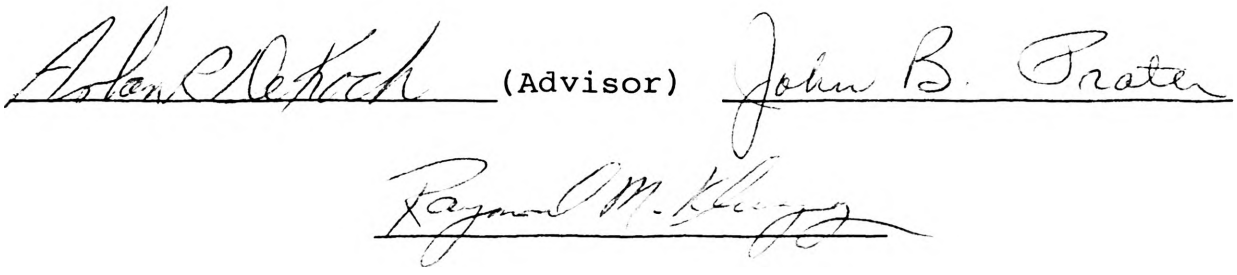
UNIVERSITY OF MISSOURI-ROLLA

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN COMPUTER SCIENCE

1984

Approved by



PUBLICATION THESIS OPTION

This thesis has been prepared in the style utilized by the ACM Computing Surveys. Pages 1-52 will be presented for publication in that journal.

A Focus of Attention Algorithm for Expert Systems

Abstract: This research is primarily concerned with increasing the performance of expert systems. A refined focus of attention strategy and its affect on performance are discussed. Early expert systems used a brute force approach to process the knowledge base. Each production rule in the knowledge base was evaluated each cycle. More recently, processing efficiency has been increased by focusing the attention of the inference engine on a subset of the rules by "filtering" for further testing, only rules that could possibly fire given the current content of the context base. Focus of attention as developed in this research increases performance over filtering systems by further narrowing the focus of attention of the inference engine, down to the subexpression level. Positive results are reported.

ACKNOWLEDGEMENT

The author is indebted to his advisor Dr. Arlan DeKock and committee members Dr. John Prater and Dr. Ray Kluczny for their invaluable guidance and encouragement. Additionally, he would like to express gratitude to Dr. Karl Kempf for technical consultation and Alan Sparks for his often used programming insights. Appreciation is also expressed to Joy Henderson and all others who lent support to this effort.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Tables Page

I.INTRODUCTION

Artificial Intelligence is a term that stands as an umbrella to a varied collection of problem solving techniques. As a sub-area of Computer Science, A.I. is not defined as much by a specific application area as it is by the class of problems it attacks. The class of problems that require the specific computer science techniques developed by Artificial Intelligence researchers may be defined as those problems where: 1) there is no "turn the crank" solutions, 2) the problem is combinatorially complex, and 3) there is no optimum answer. By way of definition, Artificial Intelligence is the branch of computer science "...concerned with creating and studying computer programs that exhibit behavioral characteristics we identify as intelligent in human behavior..." [Barr 1982].

As a quick overview, a partial and overlapping list of examples is presented of problems that require Artificial Intelligence technology and the applications that have resulted from this technology: language processing and translation, machine perception, which includes computer vision or image processing, and speech understanding, automatic programming, problem solving or planning, learning programs, game playing programs, and expert systems. The last area mentioned, expert systems, is the area of interest in this research.

Expert systems have been defined as: "... a computer program that provides expert-level solutions to important

problems ..." [Buchanan 1983a] Rule-based expert systems evolved from a more general class of computational models known as production systems. For an in depth description of "pure" production systems the reader is referred to the classical Davis paper on production systems [Davis 1975a]. The precise classification of the expert system model referred to throughout this research is a rule-based pattern directed inference system. However, it is presumed the ideas developed here are applicable generally. In fact it is difficult to exclude designs or methods on formal grounds since there is really no one formal design for current production systems and "recent implementations have explored variations on virturally every aspect, their use becomes more an issue of programming style than anything else" [Davis 1975b].

Expert systems essentially emulate an expert as that expert would apply his knowledge to a specific knowledge domain. This includes the knowledge the expert might have ascertained from books or formal training and more importantly the heuristics or "rules of thumb" the expert may have informally developed from years of experience. A wide variety of domain areas have been successfully addressed by expert systems. DENDRAL, one of the first expert systems analyzed chemical compounds. MYCIN is a medical application of an expert system which assists physicians in diagnosing bacterial infections in the bloodstream. PROSPECTER is a geological expert system that

has successfully predicted new mineral deposit locations. CAT assists with the diagnosing of diesel engines. This is but a partial list, meant to give a feeling for the diverse domains that have been attacked. Although there have been many well received expert systems developed that are in everyday use, there are some limitations and problems encountered in building and maintaining expert systems. A few problems will be discussed to put the focus of this research in context.

Extracting the knowledge from the expert has to date proven a time-consuming and error prone task. Several studies have focused on the problem but as yet knowledge engineering , as the process is called, still remains a major bottleneck in developing expert systems. There are relatively few good knowledge engineers, certainly not enough to fill the need being felt today to extract knowledge from the experts and build systems to emulate their thought processess.

Although computer power, i.e. processing speed, has experienced exponential growth in the last decade, the processing power required by expert systems demands even more powerful computers than those available to date, if they are to attack problems which require a wider breadth of knowledge than what most expert systems apply themselves to today. For instance, MYCIN performs at the expert level in diagnosing bacterial infections in the blood stream but a physician has to have first applied his expertise and

narrowed the problem down to the small domain in which MYCIN is competent before MYCIN is of any use at all. Thus, expert systems perform well within their narrow area of expertise but it is not clear that present Artifical Intelligence techniques and accompanying computing power will be able to successfully address larger problem domains and maintain the depth required to be truely functional. Given the techniques and technology in use today it seems doubtful that a "general problem solver" can be built.

An observation by Buchanan may serve to justify the experimental approach used here and put the limitations addressed in this research into a much larger context.

> " AI is still very much in the so-called `natural history' stages of scientific activity in which specimens are collected, examined, described, and shelved. At some later time a theory will be suggested that unifies many of the phenomena noticed previously and will provide a framework for asking questions. We do not now have a useful theory.
>
> Expert systems will provide many more data points for us over the coming years. But it is up to everyone in AI to do controlled experiments, analyze them, and attempt to develop a scientific framework in which we can generalize from examples. At the moment we ourselves lack the vocabulary for successful codification of our own data" [Buchanan 1981c].

A look at the architecture and operation of expert systems will be given before the specific limitations addressed by this research are discussed.

The four steps in the cycle of a pattern directed inference system (PDIS) will be stated formally, however

in this research the focus is almost entirely on the second step, thus a general description of expert systems with step one held as a constant will then be given.

Step 1:
Select a fact to begin inference with or ask the user about.

Step 2:
Find all rules that are satisfied. (Form the conflict set.)

Step 3:
Select one rule from those in Step 2. (Conflict resolution)

Step 4:
Execute this rule.(Fire it.)

The internal components of most expert systems can be broken down into three functional modules: 1) the knowledge base, 2) the inference engine, and 3) the context base to use terminology prevalent at Stanford [Davis 1975a]. Other terminology found in the literature describing the same modules refers to the context base as the working memory,scratch memory, or cache, the knowledge base as the production memory, and the inference engine as the control program.

The knowledge base (K.B.) contains production rules which in effect are small "chunks" of knowledge, usually found in the form of If-Then rules. For example:

Rule #1: IF (throat = red & temperature = 101.1)
Then symptom = flu

Rule #2: IF (throat =red & symptom = flu)
THEN sickness = pnuemonia

The K.B. may contain anywhere from 25 to 10,000 rules, the point being there is not an average or pre-set number of

rules. These rules capture and store the expert's knowledge in a form that allows the inference engine to manipulate or perform inferences on the knowledge.

As just mentioned the inference engine (I.E.) manipulates the rules found in the knowledge base in order to infer information from the information already known. For example, if it is known (in a medical diagnosing scenario) that the patient has a red throat and a high temperature, the I.E. would infer that the patient may have the flu (from Rule# 1 above). Given that the I.E. "knows" that the patient has the flu, a red throat, and a high temperature, another rule in the knowledge base may call for these conditions to be known to be true before the I.E. could infer the possibility of pneumonia from Rule #2. The I.E. then takes information known to be true and checks the rules in the K.B. to see if it can make a chain of inferences leading to a problem solution. In the above example this could be a diagnosis of the patients sickness as pneumonia.

The third component of an expert system is the context base. It was mentioned above that if a certain piece (or pieces) of information were known, the I.E. could then possibly infer other facts from the facts currently known. The way these facts are "known" by the expert system is by looking at the current contents of the context base. All facts that are "known" were put into the context base as they became known. There are two generally accepted ways in

which facts are put into the context base. First, as already mentioned, when the "IF" part of a rule is determined to be true by the I.E., that rule is said to be "satisfied" and the "THEN" part of the rule (another fact) is inserted into the context base. The astute reader might wonder how the very first facts are put into the context base to get the process started and secondly, what happens if there are no more rules that can be satisfied - given the current facts in working memory. These functions are performed by another module or component of the expert system referred to here as the user interface.

The user interface component of the expert system varies more widely from system to system than do the other three major components discussed and serves mainly as an input mechanism to the system, thus it has not been included as a major component in this discussion. As mentioned, the user interface serves to query the system user at the beginning of the consultation session, thus gaining an initial set of facts to be inserted into the context base. Secondly, when no new rules can be found to be supported by the current facts in the context base, the user interface may again be invoked to ask the user for more information. The idea being that more facts, previously not found in the context base, will be added to the context base, satisfying a rule that in turn will put another new fact into context base and the cycle will continue until a problem solution is found or the program is terminated.

To summarize then, the user interface acts to initialize the first set of facts into the context base. The inference engine then searches the knowledge base to find rules that are satisfied. A rule is said to be satisfied when every condition stipulated in the rule's "IF" part are found to be true in the context base. The "THEN" part of the satisfied rule is then entered into the context base. This cycle is then repeated until a problem solution is found.

The inference engine actually has three parts, of which only two parts have been discussed, the search and act cycles. The third part is called conflict resolution. Conflict resolution addresses the situation where more than one rule is found to be satisfied on a given cycle. When a rule is found to be satisfied, in most systems it is not immediately "fired", i.e. its fact(s) found in the "THEN" part is not immediately put in the context base, rather it is said to be put into the conflict set. The conflict set then is the set of all rules found to be satisfied - but not yet selected to be fired. Conflict resolution then selects one of the rules from the conflict set based on some criteria and fires that rule. The criteria used for selection of the rule to be fired varies from system to system. As an example, one scheme used picks the rule with the most facts in the "IF" part to be fired. There are other schemes but conflict resolution is not the focus of this research and thus will not be discussed further.

The thrust of this research, is to minimize the number of rules tested or searched in the knowledge base of an expert system by the inference engine. First, however a graph may serve useful to clarify discussion thus far and put the focus of this research in perspective. (See Figure 1.)

Expert systems hold a large promise for the future but several problems must be dealt with if they are to realize this promise. Buchanan states "The state of the art in expert systems technology is advancing, but to be quite realistic we need to look at existing limitations as well as potential power" [Buchanan 1981a]. He goes on to identify "Focus of Attention on Relevant Facts and Relations" as one of the limitations and further, cites Pople to support this observation. "As the breadth of knowledge increases, problem solvers need context-sensitive mechanisms for focusing attention on parts of the problem and parts of the knowledge base that appear most fruitful" [Pople 1977]. In conclusion he says "many methods have been tried but we have little understanding of their relative merits" [Buchanan 1981b].

Others share this concern for the need to have more sophisticated focus of attention techniques. William Mark states that "As rule-based systems grow to encompass a large number of rules, and as they are forced to work on complex knowledge structures to keep pace with modern knowledge base

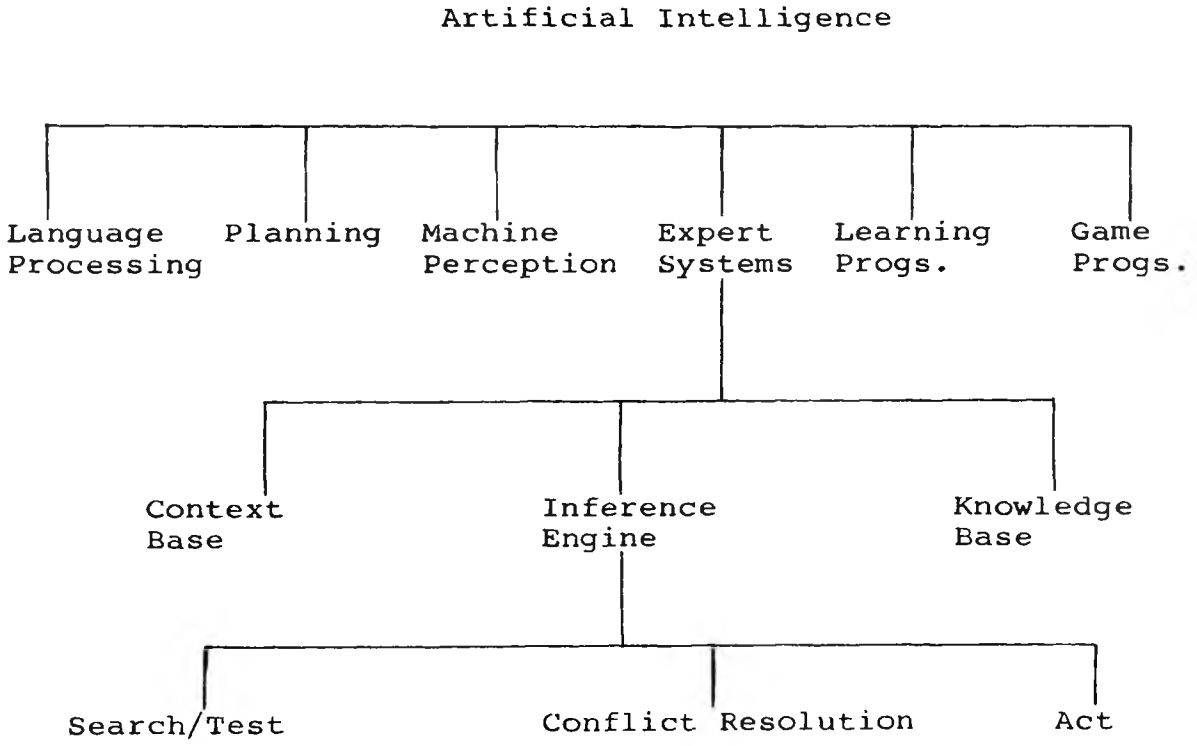


Figure 1. A Perspective on the Thrust of this Research

organizations, two problems arise..." [Mark 1980]. The first problem is particularly relevant to this research: "The inference mechanism becomes inefficient: it is hard to find the right rule to apply if there are many possibilities" [Mark 1980]. Several researchers have made significant contributions toward resolving the limitations described above. These contributions will be discussed in the next section.

Focus of attention can be broken down into at least two subproblems, the problem of organizing the context base and the problem of organizing the knowledge base. Of course these two factors interact with each other. In addressing these factors McDermott identifies the cost of determining which rules are satisfied on a given cycle as "essentially linear in the product of the number of productions in production memory and the number of assertions in working memory" [McDermott 1978a].

Here, rules are referred to as "productions", the knowledge base as the "production memory", and the context base as "working memory". Thus, as the number of elements in the context base increase and the number of rules in the knowledge base increase the cost of determining which rules are satisfied increases in an undesirable manner. Put in the light of the four steps in a cycle of a PDIS, selecting which fact or element to act upon next (Step 1), and determing which rules to test (Step 2) becomes unmanageable if the system grows too large if one uses a brute force

approach. A focus of attention is mandatory if processing is to be done in an acceptable amount of time, and if the dependency on the size of the knowledge base and the size of the context base is to be minimized.

McDermott claims that if a system has a mechanism "that enables knowledge of the degree to which each production is currently satisfied to be maintained across cycles, then the dependency on the size of working memory (context base) can be eliminated ..." [McDermott 1978a]. Additionally, he claims that if a system has a mechanism that "enables knowledge of similarities among productions to be precomputed and then exploited during a run, it is possible to eliminate the dependency on the size of production memory" [McDermott 1978a]. He indeed verifies this claim by implementing and testing the mechanisms mentioned with encouraging results.

One mechanism he used to focus on the "important" rules may be described in an oversimplified manner as being an index that contains the rule number where each element occurred. Only the rules that contain the last element inserted into the context base need to be tested,as they are the only ones that may possibly fire at this point. This information was used in conjunction with a mechanism that kept track of the degree to which each rule was satisfied across cycles. Thus, the rules that needed to be tested were further narrowed down to only those rules that contained the last element to enter the context base and

were determined to be nearly satisfied.

In an effort to be complete, Aikins has also addressed the focus of attention problem in another way. By design, her system, CENTAUR, focused its processing by recognizing patterns of data. "The overall control structure is sensitive to the initial data, and to the prototype that is being explored, which results in a more focused consultation" [Aikins 1983]. This enabled the system to divide the problem into subproblems and focus only on the rules and facts relevant within this subproblem. This type of focus of attention holds much promise and is alluded to throughout this paper. However, it is treated more as a topic for future research then a central theme in the research.

The focus of attention techniques just described eliminates much redundant testing of rules. However, the rules must still be tested. Specifically, the group of rules that are determined to be "important" i.e., nearly ready to fire, must be tested. This research focused on developing an architecture/algorithm that would enable the complete elimination of testing by carefully keeping track of the exact degree to which each rule is satisfied. By focusing on the subexpressions of each rule a more refined focus of attention was achieved that increased the efficiency of processing the knowledge base.

As a more general view, the goal of this research has been to increase the efficiency of processing the knowledge

base of an expert system. This required a novel way of organizing the knowledge base, which in turn required a refinement of present focus of attention techniques. Thus, the specific thrust of this research has been developing the refined focus of attention techniques previoulsly mentioned.

II. HISTORICAL REVIEW

Many systems have focus of attention incorporated into them. However, these have seldom (if at all) been proven to be truely domain independent. An example of focus of attention within a particular expert system will be discussed as an example of this, followed by a discussion of two efforts aimed at achieving different types of focus of attention.

The expert system MYCIN has at least two functions which interact in a manner that gives a definite focus of attention to the system. In describing future developments Shortliffe outlines [Shortliffe 1976] the possibility of "pre-screening of rules". He notes the function FINDOUT could use the LOOKAHEAD list to identify all rules referencing the parameters in their PREMISE conditions. If the condition turns out to be false FINDOUT could evaluate the relevant conditions and mark the rule as failing. Then, "whenever the MONITOR begins to evaluate rules that are invoked by the normal recursive mechanism, it will check to see if the rule has previously been marked as false by FINDOUT" [Shortliffe 1976]. Although this is not a particularly refined focus of attention, the point is that many systems have some means of reducing the number of rules that must be tested each cycle. They have some means of focusing the attention of their inference engine or they would undoubtedly not be "high performance" expert systems.

Another expert system that has focus of attention

"hardcoded" into it is Aikin's redo of PUFF, named CENTAUR [Aikins 1983]. CENTAUR utilizes frames or "prototypes" along with production rules to improve the performance of the system. The initial data along with a knowledge of the prototypical situations are used to select the subproblem the present consultation most closely fits, thus giving the remaining consultation a more specific context within which to work.

Traditional expert systems have not taken into account the implicit groupings of rules according to Aikins. The modularity of the rules in pure rule-based systems prevents organizing the knowledge base in a manner that could partition groupings of similar rules. The partitioning of the rules in CENTAUR allows the system to focus on only those rules that are relevant at a given time.

The frames in CENTAUR are used to guide rule selection by focusing the search for new information and asking for only the most relevant information from the user. This is an obvious example of "Step 1" type focus of attention, organizing the context base. Within the frames, there is a set of slots. The slots "provide an explicit `place' for information in the frame so the system can better judge when enough inforamtion is known to determine a solution for the problem" [Aikins 1983]. This is not totally dissimilar from McDermott's reference count which will be discussed further momentarily. The point here is that the system architecture can be designed to implicity focus the consultation and can

be useful in determining which fact needs to be asked next and if a problem solution has been found. The focus of attention techniques described here might well be applicable generally but for now are embedded in a specific system.

As was mentioned in the previous section, McDermott has addressed the focus of attention problem with positive results. McDermott's techniques center around organizing the knowledge base, i.e. deciding which rules should be considered in a given cycle [McDermott 1978].

In developing his techniques he identifies two main "knowledge sources" that can be taken advantage of to reduce the cost of finding the conflict set. The "pre-execution" knowledge source he describes consists of the knowledge about which elements or conditions occur in which production rules. The "during-execution" knowledge source consists of knowledge about which elements are supported by the context base, i.e., within a rule, which of its elements are presently known to be true in the context base. (Another knowledge source was discussed briefly but not utilized in the program implementation.)

The knowledge sources described above are used to set up "filters". These filters admit for further testing only those rules that are likely to be satisfied. As an example, if `A' was the last element to enter the context base, only those rules containing `A' would be tested. They are the only rules that could become satisfied when `A' enters the context base.

A refinement of this focus of attention involves the use of another filter. This filter works in conjunction with the first filter described by maintaing a reference count based on the number of elements in each rule that need to be known to be true, before the rule can be satisfied. For example, as an element enters the context base the reference count for all rules that contain this element in their antecedent are decremented by one. When the reference count of a rule decrements to zero, the rule is identified as a candidate that is possibly satisfied and is tested.

Thus, McDermott utilizes different knowledge sources and filters to focus the attention of the inference engine on only the rules that are likely to be satisfied. This is primarily aimed at increasing the efficiency of step 2 in the PDIS cycle, organizing the knowledge base.

In summary, several systems have focus of attention techniques "built in". MYCIN utilizes knowledge about which rules have failed to prevent it from re-evaluating a rule that has failed. CENTAUR focuses the attention of the inference engine to pre-specified subproblems by being sensitive to the initial data. Finally, McDermott has pointed out knowledge sources one can take advantage of to focus the attention of the inference engine on only the rules that are likely to be satisfied on a given cycle.

III. THEORETICAL VIEW

In this section a theoretical view of techniques that would optimize the focus of attention of the inference engine will be discussed. The discussion briefly describes the macro-level techniques that could be used and then centers around the micro-level techniques. Macro-level is referred to here as those techniques that organize the knowledge base into partitions or subproblems, before the actual inferencing to a problem solution is done. Micro-level is referred to as those techniques that optimize the inferencing itself. It is quickly acknowledged that these classifications overlap and work in conjunction with each other, but the "spirit" of the classifications may serve to clarify the domain being discussed.

An optimal macro-level focus of attention technique would narrow the entire knowledge base down to a "working knowledge base", i.e. only a subset of the knowledge base. This subset would need to contain all rules that might be used during this consultation but not contain any rules that the user, from the outset, can identify as being irrelevant to this particular consultation. For example, if there was an expert system which had a knowledge base large enough to diagnose all possible diseases but the user could narrow the disease down to either being a heart or lung disease from the outset, only the rules that relate to heart or lung diseases need be used during this consultation. At the present time one of the limiting factors to having a

knowledge base of this size, is the lack of techniques to focus the attention of the inference engine on the relevant rules. However by "chunking" the knowledge base in the manner described above this problem could possibly be reduced to a manageable situation. This would allow faster processing to a problem solution by focusing on only the set of rules relevant at a given time. Further, this would allow a more directed and focused asking of questions by the system. Thus, both step 1 and step 2 of the PDIS cycle would be made more efficient and focused.

An optimal micro-level focus of attention would use all possible knowledge sources to focus on only the rules that are satisfied. This would require preserving information gained from past cycles, in a manner that would incur minimal processing cost. Certainly the cost of utilizing the technique must not exceed the gain realized from the added focus of attention, and in fact unless the cost was considerably less, one would question whether developing and using the technique would be worth the time and effort at all.

The minimum micro-level focus of attention technique involves focusing on only rules that can possibly fire. That is, any rule that contains a negated element of an element found in the context base should be marked in some manner so that the rule is never evaluated, as it cannot fire. Additionally any rule that has already fired should be marked so that it will not be evaluated again.

A more refined technique requires a knowledge source that contains information about which rules contain each element. Thus, as an element enters the context base the system could narrow down immediately the set of rules that may fire, i.e. the rules that contain that element in their antecedent. A further refinement would require a mechanism to monitor the degree to which each of the rules in the above subset were satisfied. This would allow the testing of only those rules that were determined to be "probably" satisfied.

If the system was additionally augmented with a technique that allowed monitoring of the rules at the subexpression level, it could monitor the degree to which each rule was satisfied in a more precise manner. For example, in the expression "(A and B) or C implies D" it is clear that at most, two of the elements found in the antecedent of this rule must enter the context base before the rule is satisfied. If the first element is "C", then only one element in the antecedent of the rule is needed to satisfy the rule. Thus, a knowledge source that contained information on the exact degree to which the subexpressions within the rules are satisfied, coupled with knowlege about the logical relationships between the subexpressions, could eliminate testing altogether.

The last described technique would require a way to preserve the history of previous cycles. It would need a mechanism that updated a reference count for each

subexpression within each rule as an element contained in the subexpression was put into the context base. This would preserve the history of previous cycles and eliminate the need to repeatedly retest each element when one of the other elements contained in the subexpression was inserted into the context base.

In summary, the optimal focus of attention technique would require a means to partition the knowledge base into a "working knowledge base". Then within this working knowledge base, the technique could focus on only the relevant rules, and within the set of relevant rules, focus on the number of elements each subexpression needed and the relationships between the subexpressions. If done carefully, this would eliminate the need to test the rules altogether, as the system could be signaled that a rule was satisfied when the required subexpressions within the rule became satisfied. The problem with these techniques has previously been the ability to do this in a cost effective manner. The techniques developed in this research and their ability to achieve the above in a cost effective manner will be discussed in the remaining sections.

IV. DESIGN AND IMPLEMENTATION

A. OVERVIEW

As related to the four steps in the cycle of a PDIS as outlined on page 4, this research is particularly aimed at optimizing step 2, providing an improved technique for organizing the knowledge base. More precisely, this research is aimed at developing a system architecture that increases the efficiency of processing in an expert system by reducing the number of rules that must be tested on a given cycle.

This goal is restated to accent the rationale behind the implementation. As this research is more concerned with step 2 than with step 1 of the PDIS cycle, the facts were given to the systems in an arbitrary order. The entire focus of attention problem of which fact is the best fact to ask about or infer from next was ignored. The facts were inserted into the context base in an intentionally uninteresting sequence, so as not to influence the focus of attention problem that is the thrust of this research. Thus step 1 of the PDIS cycle for all three systems implemented in this research amounted to reading one fact at a time from data files.

The implementation of the ideas set forth earlier in the paper will now be discussed in some detail. First the components of the three systems will be identified, then the three inference engines will be described.

All three systems can be broken down to two basic

components: 1) Genbase and 2) an inference engine. Genbase generates the knowledge base that the three inference engines, Brute4s, Rulefocus, and Subexfocus manipulate. They all share the same basic data structures, with only a few anomalies that will be discussed. Genbase reads in the production rules one at a time from a text file and compiles each rule into a tree structure. (See Figure 2.) The pointer to the head of this tree is stored in an array. Thus when all rules have been compiled, the knowledge base is actually a collection of tree stuctures, with the head node of each of these trees stored in an array called "ruleset".

The rules read from the text file can be any arbitrary expression of the form: "Antecedent => Consequence". The antecedent is composed of variables logically connected by any combination of ands(*), ors(+), and nots(~), nested within any number of matching parentheses. The "=>" is an implication sign, which is followed by a list of variables separated by commas which represent the consequences. The Genbase module can process any depth of nesting of parentheses and any number of variables on the antecedent side and any number of variables listed on the consequence side. The consequence side does not allow for logical expressions of the type " Antecedent => A or B".

The author has imposed limits, as general guidelines, such that the rules actually used consist of one to four

RULE 1: (A AND B) OR C IMPLIES D

IN TEXT FILE: (A * B) + C => D

RULE 1 CONVERTED TO DATA STRUCTURE FORM:

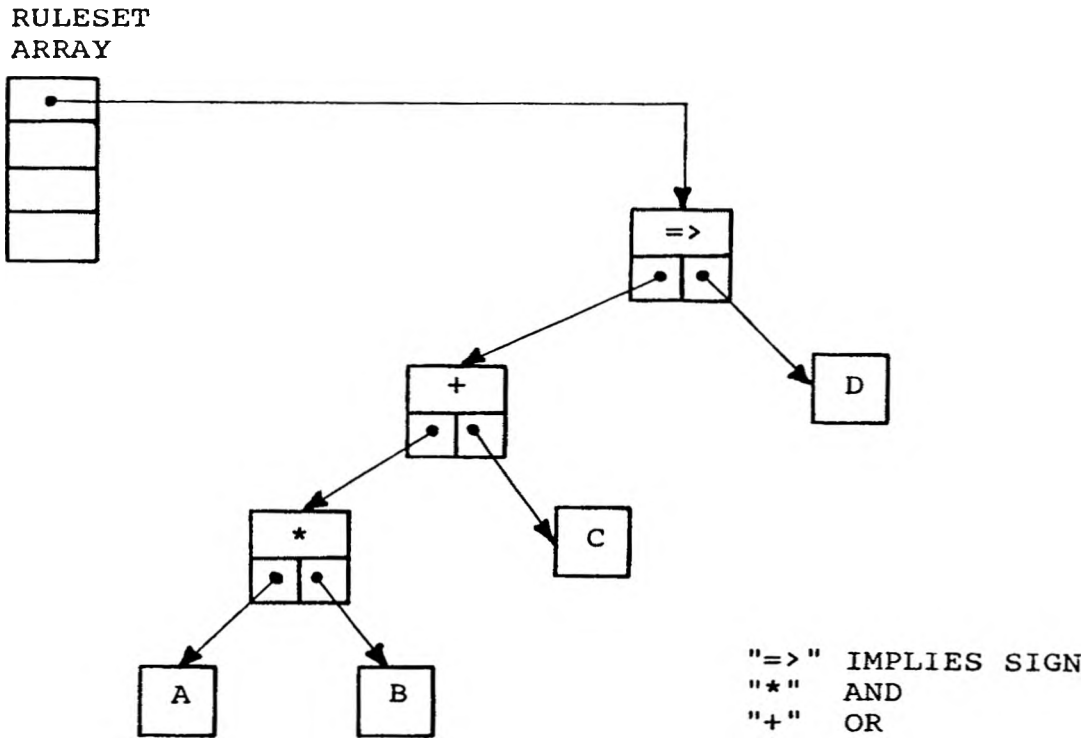


Figure 2. Rule 1 in Data Structure Form.

variables on the antecedent side and one variable on the consequence side. This is felt to be a realistic representation of production rules as they are found in expert systems today.

To summarize, Genbase takes a file of rules from a text file and constructs the knowledge base into the form of an array of tree stuctures. Simply, Genbase is one form of a rule compiler that utilizes simple recursive-descent parsing techniques to process the rules into data structures.

To implement the algorithms discussed earlier, the tree structures need to be in a slightly different form. The procedure called "Normalize" accomplishes that. It restructures the tree so that all "nots" are propagated down to the bottom level or the leaves of the tree. Utilizing DeMorgan`s Law [Manlo 1972] all "nots" found in or before expressions are forced down to the variable or fact level. This allows for more efficient processing in the Brute4s inference engine and is central to the building and manipulation of the major indexes critical to the high performance level achieved by the Subexfocus inference engine.

Three systems are implemented in this research to compare the increase in efficiency achieved when the techniques described earlier in this paper are used to focus the attention of the respective inference engines on first all the rules (Brute4s), then a subset of the rules (Rulefocus), and then the subexpressions (Subexfocus). The

first inference engine, Brute4s (brute force), provides a benchmark. It searches the entire knowledge base and evaluates all rules that have not already been fired on each cycle. Inference engines in early expert systems did in fact use this type of exhaustive searching and matching.

The Rulefocus inference engine demonstrates one of the techniques developed by McDermott [1978]. Rulefocus builds an index before execution that contains for each fact a reference to the rules containing that fact. Rulefocus then exploits this information during execution by evaluating only the rules listed in the index for the last fact to be entered into the context base. The Subexfocus inference engine demonstrates an advance in design developed in this research which increases the efficiency of processing the knowledge base to a problem solution by focusing on subexpressions of rules, rather than on rules.

Several of the program procedures are used by all three inference engines. For example, they all use the same procedures to 1) insert a rule found to be satisfied into the conflict set, 2) perform conflict resolution, 3) detect a problem solution, 4) insert the consequences into the context base and 5) obtain facts from the user. Thus all three programs use the same knowledge base and identical context base schemes. As a footnote, the conflict resolution used was a recency first scheme. The consequence of the most recent rule to be inserted into the conflict set for each cycle was inserted into the context base.

B. BRUTE FORCE INFERENCE ENGINE

The Brute4s inference engine, as mentioned, evaluates each rule to determine if it is satisfied, given the facts in the context base at that time. The procedure "User_interface" is responsible for the initial insertion of facts into the context base to get the process started. The user is queried as to the facts he wishes to insert into the context base. These facts are then inserted into the context base one at a time. When a rule is found to be satisfied by the procedure "Bruteval", it is inserted into the conflict set. After each rule has been evaluated via one pass through the knowledge base, (this is referred to as one cycle) the conflict set will consist of the set of all rules found to be satisfied at that point.

The procedure "Conflict_resolution" is then called to select one rule from the conflict set to be fired, i.e. to insert the consequent of the selected rule into the context base. Cycle follows cycle until either 1) a problem solution is found or 2) the conflict set becomes empty. Any time a problem solution is found, it is printed out and the program is halted. If the conflict set becomes empty before a problem solution is found, User_interface is called to insert another fact into the context base. The cycle is then repeated until a problem solution is finally found.

C. RULEFOCUS INFERENCE ENGINE

The Rulefocus inference engine utilizes procedures from

the other two inference engines. It uses the same data structures and evaluator as does Brute4s. (See Figure 2.) The difference is that it does not evaluate every rule, every cycle rather it evaluates only the rules that contain the last fact entered into the context base.

The index it utilizes to guide it to these rules is constructed in the same manner as the index for Subexfocus. That is, as the data structures are being built, a fact list is also built. The difference is that the fact list references rules rather than subexpressions. The data structures are not restructured with location pointers built to point to the restructured operator nodes, rather the rule number in which this element occurred is stored in the location node(s) for each fact.

Thus as a fact enters the context base, the fact list is searched to find the fact, then the rule list for this fact is traversed. The same evaluator that is used in Brute4s is then called for each rule number contained in the rule list. In effect, all rules that contained this fact are evaluated to see if they are satisfied at this point in the consultation. As a rule is found to be satisfied it is inserted into the conflict set and the cycles continue in a manner identical to the other two inference engines.

D. SUBEXFOCUS INFERENCE ENGINE

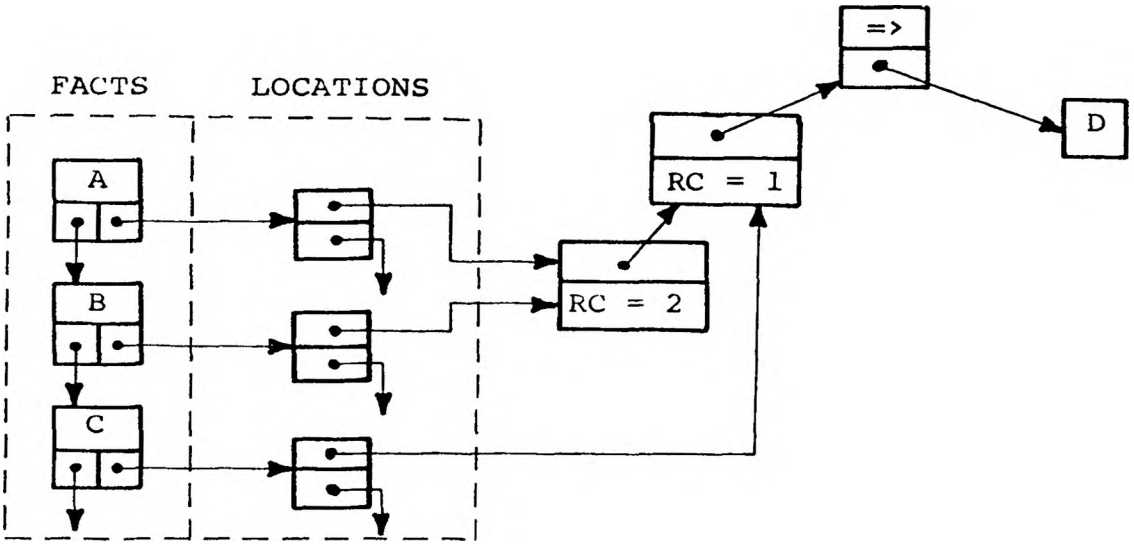
The Subexfocus inference engine requires a special data structure, different from the data structures described so

far. (See Figure 3.) The procedure "kwrestu" (restructure) takes the data structures after they have been processed into a normal form by the "normalize" procedure and restructures them into a form that allows the Subexfocus to evaluate them. The restructuring is performed in a recursive descent manner one rule at a time for each rule in the knowledge base as follows.

Before restructuring occurs, the head node of the data structure is always the node containing the implication sign "=>" and two pointers -the right child and the left child. The right child points to a node that contains a consequence, which in turn may point to another consequence if there are two facts to be asserted from this rule. The left child points to the node containing either a fact in the antecedent or a binary logical operator that in turns points to other operators which point to facts contained in the antecedent part of the rule.

The restructure process starts by building a new data stucture from the implication node. The new data structure contains two fields. The first field contains the information that this node is an implication node. The second field contains a pointer to the consequences, which are left exactly as they were. The restructure procedure keeps a pointer to this head node and starts to recursively process down the left side of the tree.

(A * B) + C => D RESTRUCTURED FOR SUBEXFOCUS INFERENCE ENGINE



RC = REFERENCE COUNT

Figure 3. Rule 1 Converted to Subexfocus Form.

As the antecedent part of the tree is restructured, if it finds that the node is an operator, it builds a node that contains a reference count and a pointer which points back up to the node above it in the tree. The reference count is the number of facts that must be found to make this expression true. For instance an "and" node (which points to exactly 2 nodes) must always have exactly the two nodes below it true before it is true itself. "And" nodes always have a reference count of two. If the operator node is an "or", then the reference count must always be one. These reference counts follow from the boolean algebra definition of conjunctive and disjunctive expressions which states that only one of the conditions or facts of an "or" expression is required to be known true for the expression to be true, whereas all facts of an "and" expression must be known to be true before that expression is considered true.

In the antecedent part of the rule, if the node is not an operator then it must be a "variable" or fact. When the restructure process finds a variable it creates a new node that contains a pointer to the location of the node that is directly above the variable in the tree, i.e. the operator node of which this variable is a fact or condition. This fact is inserted into a forward link list that acts as an index. Each node in this forward link list contains a pointer to the node that contains the operator to which this variable is a condition. Thus variable "A" has a link list associated with it that contains node(s) that point to

all the subexpressions which have the variable "A" as a condition. Each variable that occurs in the knowledge base has a list of nodes that point to all the subexpressions or operators of which this fact is a condition if the subexpression is to be true.

An index of this type has been used in previous expert systems [McDermott 1978]. The distinguishing feature of this index is that it does not contain the references to rules in which each variable occurs; rather it contains a reference pointer to the exact subexpressions in which each variable occurs.

The Subexfocus inference engine is identical to the Brute4s inference engine relative to the user interface, conflict resolution, and notification of problem solution. The difference between the two lies in the evaluator. The evaluator for Subexfocus takes advantage of the restructured data structures in the following manner. As a fact is inserted into the context base, the procedure "kweval" is called. This procedure indexes into the table described above on the name of the fact. It then begins to traverse the list of references to subexpressions where the fact occurs. Each reference to a subexpression is implemented as a reference to the appropriate operator node which contains the expression's reference count.

Decskp0 (decrement and skip on zero) first checks to see if this is an operator node or if it is an implication node. If it is an implication node the rule is assumed to

be satisfied and the corresponding consequence is put into the conflict set. If it is not an implication node then it must be an operator node and the node`s reference count is decremented by one. If this decrement brings the reference count to zero, the subexpression has been found to be true and decskp0 recurses, this time with the pointer to the node above this operator node. The procedure will recursively call itself climbing up the restructured tree until it finds either an operator node that has not been satisfied, i.e. its reference count does not equal zero, or an implication node is found signaling that the rule has been satisfied.

V. METHODS AND PROCEDURES

The metric used to measure the efficiency of the three different architectures/algorithms and the conditions surrounding the application of the metric will now be addressed.

The experiment performed here was aimed at testing and benchmarking the focus of attention techniques previously described. Performing tests on a synthetic knowledge base has hopefully removed any favoritism toward one of the three systems tested that might occur in a non-synthetic knowledge base. This was intended to either filter out or hold constant secondary factors that might affect the statistics ascertained from the experiments.

The metric used then was the number of nodes visited. One node visit was counted when the inference engine visited any of the following: 1) an implication node, 2) an operator node or 3) a variable node. The implication node is the head node in the data structures. An operator node is either an "AND" node or an "OR" node. A variable node is a node containing a fact, also called an element.

The metric should magnify the differences in the three engines. As Brute4s repeatedly and redundantly evaluates all subexpressions of each rule it would be expected to have by far the largest number of nodes visited. Similarly, as Rulefocus may re-evaluate all subexpressions within a rule several times before the rule is found to be satisfied, it would be expected to visit more nodes then Subexfocus. As

Subexfocus avoids redundant and repeated evaluations by preserving the history of previous cycles, it would be expected to visit the least number of nodes. Thus, the metric does indeed accent the differences in the three systems and provide a measure of the quality of focus of attention each system provides.

Execution time was not used as a metric because it was felt clever programming could easily bias one of the methods. For instance, if the fact list and the location list were hashed instead of sequentially searched, the execution time of the Subexfocus system could have been reduced considerably. The metric was chosen to accent the technique developed as opposed to the programming cleverness.

The conditions of the experiment were designed to provide an unbiased environment in which to test the three systems. All three systems ran on the same knowledge base with the same sets of facts. Additionally, the three systems used the same procedures for everything except the procedures that guided the search of the knowledge base, as this is the focal point of the experiment.

The knowledge base was derived from an acyclic-directed graph. The graph was drawn with 22 nodes across the top of the graph that branched down through 6 layers of intermediate nodes to 10 terminal nodes. The top 22 nodes represented initial observations, the intermediate nodes represented inferences made from the initial observations

and the 10 terminal nodes represented the 10 goals.

From the graph, 117 rules were derived with 104 inferred facts. The goals required between 1 and 14 initial facts to reach the 10 different goals.

By tracing back up through the graph from each of the goals, the minimum number of initial facts required to reach each goal was determined. Thus, there were 10 sets of facts, one set for each goal. These sets were put into separate data files.

As was detailed in the design and implementation section, each system began execution by asking for a fact and then inserting it into the context base. The inference engine then inserted any satisfied rules into the conflict set and performed a conflict resolution. This was repeated until a goal was found or the conflict set was empty. If the conflict set became empty the user was asked for another fact and the process was repeated.

One run of the experiment consisted of putting one set of facts that led to one of the goals into a file and directing the inference engine to read from the file when a fact was needed from the "user". Each of the three systems was run on each of the 10 goals. The number of nodes visited were counted for each of the 30 runs. The results are presented in the following section.

VI. RESULTS

Table I presents a comparison of search method by goal based on the number of nodes visited. The goals in the table are listed in a manner such that goal 1 is the goal that required the least number of initial facts and goal 10 is the goal that required the largest number of initial facts. The average number of nodes visited for the three inference engines for the 10 goals are as follows: 12,384 for Brute4s, 328 for Rulefocus, and 92 for Subexfocus. (See Table 1.)

Table II presents this same information in ratio form. Complexity of goal, as measured by the minimum number of initial facts required to find each goal, did not affect the relative frequencies for the three engines.

Ratios were calculated by dividing the number of visits of each entry in the table for Brute4s and Rulefocus by the number of visits in the corresponding row for Subexfocus. The equation is "number of nodes visited/number of nodes visited for Subexfocus". (See Table II.)

The Brute4s system required on the average 145.8 times more node visits then the Subexfocus system. Rulefocus required about 3.6 times more node visits then Subexfocus.

TABLE I

A COMPARISON OF THREE INFERENCE ENGINES BY GOAL BASED ON NUMBER OF NODES VISITED

GOAL	BRUTE FORCE ENGINE	RULEFOCUS ENGINE	SUBEXFOCUS ENGINE
1	1313	33	10
2	3510	81	21
3	7501	185	49
4	11736	318	83
5	13240	281	79
6	16487	431	118
7	14587	408	109
8	21719	536	151
9	22340	586	161
10	21313	525	155
AVG.	12384	328	92

TABLE II

A RATIO COMPARISON OF SUBEXFOCUS TO BRUTE FORCE AND RULEFOCUS BASED ON THE NUMBER OF NODES VISITED

GOAL	BRUTE FORCE ENGINE	RULEFOCUS ENGINE
1	131.3	3.3
2	167.1	3.8
3	153.1	3.7
4	141.4	3.8
5	167.6	3.6
6	139.8	3.6
7	133.9	3.8
8	143.8	3.5
9	140.5	3.7
10	138.8	3.3
AVG.	145.8	3.6

$$\text{RATIO} = \frac{\text{NUMBER OF NODES VISITED}}{\text{NUMBER OF NODES VISITED BY SUBEXFOCUS}}$$

VII. CONCLUSION

The statistics were consistent throughout and do indeed support the premise on which the techniques developed in this research were based; namely, that a refined focus of attention may be achieved by focusing on subexpressions and that this refinement can reduce the cost of finding the conflict set, by reducing the number of nodes the system has to visit. Roughly, Brute4s visited nearly 150 times more nodes then did Subexfocus, while Rulefocus visited over three times as many nodes as Subexfocus.

Brute4s was primarily used to give a benchmark to be compared with and to be improved upon. The only knowledge source that Brute4s avails itself of when searching the knowledge base and testing each rule is the knowledge of which rules have already fired. The Rulefocus inference engine on the other hand lives up to its name and focuses only on the rules that contain the last element that entered the context base by availing itself of the pre-execution knowledge of which rules each element occurred in.

The Subexfocus inference engine also lives up to its name by focusing on the subexpressions within each rule. It avails itself of more precise knowledge. For insight the simplicity of the implementation of the updating process will be shown here. The updating process in Subexfocus centers around the procedure decskp0 (decrement and skip on zero). The core of this procedure is given to accent its simplicity.

```
Procedure decskp0 ( subexpression);

      reference count = reference count - 1;

      if ( reference count = 0 ) then
      call decskp0 (next subexpression);

end procedure;
```

The efficiency argues well for the ability of the Subexfocus system to achieve many of the goals set forth in section III in a cost effective manner.

The results follow from and verify the theoretical view of optimizing search efficiency as discussed in Section 4. Brute4s has practically no focus of attention technique built into its searching process. Rulefocus utilized knowledge about which elements occurred in which rules and greatly improved the focus of attention of the inference engine as evidenced by its large reduction in number of node visits. However, it did have to re-evaluate rules often enough to lose an appreciable amount of effeciency.

The Subexfocus system demonstrated an improvement of design. By a further refinement of focus of attention, i.e. focusing on the subexpressions and preserving the history of previous cycles, it avoided ever re-evaluating any subexpressions or rules. Thus, it accomplished one goal of the research and demonstrated a techniqued that reduces the number of node visits. The reduction in the number of the node visits in turn, reduces the cost of finding the conflict set which increases processing efficiency.

Further, it is the author's speculation that as the

average number of elements in the antecedent increased from three (the average of the knowledge base used) to say, eight, the Subexfocus system would show a more pronounced superiority over systems such as Rulefocus. The rationale being that if a rule contains eight elements in the antecedent, all connected by "AND"s, a technique such as the one used in Rulefocus would evaluate the rule seven times more then neccessary. That is, as each of the eight elements in the rule's antecedent entered the context base, the rule would be evaluated. This would be done needlessly because the rule would not be satisfied until the last required fact entered the context base. Thus the system would have had to visit eight varible nodes plus seven operator nodes eight times, for a total of 120 node visits before the rule was determined to be satisfied.

A Subexfocus scheme would require only eight node visits. Admittedly, this is a worst case scenario but the point is that this knowledge base did not bias the results in favor of the Subexfocus system.

Throughout this research the number of nodes visited has been the critical metric for reasons previously stated. However, execution time will be the ultimate criteria for the techniques developed here. Subexfocus could be improved, from the point of view of execution time, by avoiding a linear search of the fact list. This data structure could be hashed for improved performance.

Other improvements and variations on the system and techniques developed in this research are discussed as ideas for further research in the next section.

VII. FURTHER RESEARCH

As the Subexfocus expert system was developed mainly as a prototype to test the specific focus of attention techniques described throughout this paper, it has numerous possibilities for improvement and further research.

Perhaps the most obvious place for improvement is the conflict resolution scheme. Two alternate schemes that would be easy to implement are suggested. The first scheme would be to pick as the rule to fire, the one which has as its consequent that fact which is in the antecedent of the greatest number of other rules. For example, if there are two rules in the conflict set and one is to be chosen to fire, i.e. its consequent is to be put in the context base, go to a precomputed index that contains information on which of the two consequences will help satisfy the greatest number of other rules, and choose the rule that, if fired, will satisfy the most other rules.

Another scheme, not totally dissimilar, would be to scan all reference counts in the system and then pick the rule, whose consequent would satsify either 1) the most number of rules or 2) the "most important" rule. The principle difference between this scheme and the former is that the reference counts would be more recent information then the precomputed index . The index could also be updated, but this would incur considerably more processing cost.

Determining the "most important" rule is indeed a topic

in and of itself. One idea will be presented. To select the most important rule one could start at the goal nodes and chain backward through the antecedents checking for any of the consequences of the rules in the conflict set. The first consequence of a rule in the conflict set that was found as an antecedent to a goal, or an antecedent to a goal's antecedents, would be picked to be inserted into the context base. As an oversimplified example, "A and B implies Goal", and "A" was in the context base, it would be a judicious choice to pick the rule in the conflict set that had as its consequence, "B".

Another possible scheme for future research, not pertaining to conflict resolution, would amount to a depth first search. Once a rule has been satisfied, fire it immediately by allowing the decskp0 (decrement and skip on zero) recursive procedure to be invoked. The cascading effects from this sort of technique might be interesting.

Another permutation that could be implemented would allow "What if?" sessions. A fact could easily be retracted by simply going back and incrementing all previously decremented reference counts affected by the fact. Thus in an interactive session, one could give it a fact, evaluate any results from the addition of this fact, and then retract the fact if results were not desirable.

Parallel processing is another addition to the system that may expedite processing. When traversing the location list, each pointer to an operator node on the list could be

sent off concurrently to do its processing with the decrement and skip on zero procedure.

The last item for further research relates to meta-conclusions. Meta-conclusions would be most closely related to Aikin's research dealing with frames. The most obvious difference is that meta-conclusions would provide domain independent high level focus of attention, as opposed to domain dependent.

One could precompute indexes on all the rules that were involved in each of the goals line of inference. This could be done by chaining back up through the antecedents of each goal to the initial facts. Then if the user could narrow down the goal possibilities by indicating any uninteresting or implausible goals, the knowledge base could be reduced to contain only the union of the sets of rules neccessary for the remaining goals. This "working knowledge base" would then be smaller and hopefully a bit more focused to the problem being pursued.

This would in effect partition the knowledge base into "chunks" similar to Aikin's prototypes. Given the structures used in the Subexfocus system, this would not require many alterations.

To summarize, further study could be done to improve the expert system design developed in this research. Conflict resolution, "What if?" games, parallel processing, and meta-conclusions are all fertile areas for further research.

BIBLIOGRAPHY

Aikins, J. S. 1983. Prototypical knowledge for expert systems. Artificial Intelligence 20,2, 169-210.

Barr, A. 1982. Artifical Intelligence: cognition as computation. Rep. Stan-CS-82-956, Dept. of Computer Science, Stanford Univ., Stanford, Calif., pp. 1.

Buchanan, B. 1981a. Research on expert systems. Rep. Stan-CS-81-837, Dept. of Computer Science, Stanford Univ.,Stanford, Calif., pp. 5.

Buchanan, B. 1981b. Research on expert systems. Rep. Stan-CS-81-837, Dept. of Computer Science, Stanford Univ.,Stanford, Calif., pp. 11.

Buchanan, B. 1981c. Research on expert systems. Rep. Stan-CS-81-837, Dept. of Computer Science, Stanford Univ.,Stanford, Calif., pp. 19.

Buchanan, B. 1983. Principles of rule-based expert systems. Advances in Computers 20, pp. 164.

Davis, R. and J. King 1975a. An overview of production systems. Rep. Stan-CS-75-524, Dept. of Computer Science, Stanford Univ.,Stanford, Calif.

Davis, R. and J. King 1975b. An overview of production systems. Rep. Stan-CS-75-524, Dept. of Computer Science, Stanford Univ.,Stanford, Calif., pp. 1.

Manlo, M. 1972. Computer logic design. Prentice-Hall, Inc.,1972, pp.39.

Mark, W. 1980. Rule-based inference in large knowledge bases. First annual conference on artifical intelligence proceedings, 1980, pp.190.

McDermott, J., Newell, A., and Moore, J. 1978. The efficiency of certain production system implementations. In Pattern-directed inference systems, edited by Waterman, D.A and Hayes-Roth, F. Academic Press, New York , 1978, pp.155-176.

Pople, H.E. 1977. The formation of composite hypotheses in diagnostic problem solving - an exercise in synthetic reasoning. Proceedings IJCA-77, pp. 1030-1037.

Shortliffe, E.H. 1976. Compute-based medical consultations: mycin. American Elseveir Publishing Co., Inc., 1976, pp. 153.

VITA

Kevin Wayne Whiting was born November 10, 1955 in Great Bend, Kansas. He received his primary education in Dighton, Kansas and secondary education in Newton, Kansas. He has recieved his college education at University of Nevada-Las Vegas, Las Vegas, Nevada, Kansas State University, Manhattan, Kansas, and the University of Missouri-Rolla, Rolla, Missouri. He received a Bachelor of Science degree in Psychology from Kansas State University in Manhattan, Kansas in May 1982.

He has been enrolled in the Graduate School of the University of Missouri-Rolla since September 1982.