PROCEDURAL EXPERT SYSTEMS

Michael Georgeff,
Artificial Intelligence Center,
SRI International,
333 Ravenswood Ave.,
Menlo Park, Co., 94025.

Umberto Donollo,
Department of Computer Science,
Monash University,
Clayton, Victoria, 3168,
Australia.

Abstract

A scheme for explicitly representing and using expert knowledge of a procedural kind is described. The scheme allows the *explicit* representation of both declarative and procedural knowledge within a unified framework, yet retains all the desirable properties of expert systems such as modularity, explanatory capability, and extendability. It thus bridges the gap between the procedural and declarative languages, and allows formal algorithmic knowledge to be uniformly integrated with heuristic declarative knowledge. A version of the scheme has been fully implemented and applied to the domain of automobile engine fault, diagnosis.

§1 Introduction

Expert systems have been very successful in a wide variety of applications [Michie 1980]. The purpose of these systems is to acquire the formal and heuristic knowledge of specialists in some particular domain, and then to use this knowledge to solve problems that the specialists would normally tackle. The domains in which expert systems have proved most successful are those where heuristic knowledge is important, either because the related problems are ill-defined or are too complex for purely formal methods to be tractable [Buchanan and Duda 1982).

Some of this specialist or expert knowledge is best expressed as a set of facts and axioms or rules about those facts. Tor example, here is some declarative knowledge from the automotive domain [Sully and Unstead 1978]:

What faults reduce gasoline flow from the pump?
A choked suction line or dirty filter.
A cracked diaphragm.
Dirt under a valve.
Leakage from the pump or discharge line.

Rule-based expert systems seem to model this aspect of expert knowledge quite well. Here is a typical rule for one of the above faults :

> IF there is reduced gasoline flow in the fuel pump
> THEN CONCLUDE that the suction line may be choked.

The success of these rule-based expert systems stems largely from the fact that the knowledge representation is appropriate to the problem domains that they address. Indeed, the authors usually stress the importance of choosing the "right" domain [Buchanan and Duda 1982].

On the other hand, some of our knowledge about solving problems may be viewed as procedural knowledge that describes *sequence** of things to do or goals to be achieved. This kind of knowledge is often difficult and cumbersome to describe in a declarative manner [Winograd 1975]. In fact, even though some procedural knowledge can be incorporated in pure rule-based systems, it is there only because the rule interpreter executes the rules procedurally in some specified order. This means that procedural knowledge and the establishment of contexts in which a particular inference is valid can only be represented implicitly in the system (e.g., by ordering the clauses of a premise and thus ensuring a particular sequence of evaluation).

This can create dependencies and interrelationships that tend to make the knowledge base not quite as modular or flexible as perhaps was originally intended. Because of the homogeneity of the rule representation, it is not possible to distinguish between those rules for which the order of invocation is important and those for which it is not. This is not only bad methodology, but it impairs the explanatory capability of the system and reduces the possibilities for efficient implementation on multiprocessor machines.

Note that we are not saying that such knowledge *cannot* be represented declaratively — all we are saying is that in some domains it cannot *easily* or *naturally* be so represented, which complicates the construction of the expert system and reduces thr explanatory capability of the system (see [Clancey 1981] for a more detailed discussion of these points).

These problems have encouraged some researchers to investigate ways of representing procedural knowledge explicitly. In the simplest cases, special mechanisms are introduced, such as heterogeneous sets of rules that are dissimilar in nature from the rest of t he knowledge base (e.g., the therapy rules in MYCIN [Shortliffe 1970]), or precedence relations, which require that certain rules be invoked before others (e.g., "contexts" in Prospector [Reboh 1981]). Alternatively, specialist procedures may be used for certain sections of the problem-solving process. However, these approaches are either not sufficiently general to express much procedural knowledge, or tend to destroy many of the desirable properties of the system, such as explanatory capability, modularity, and, most importantly, the ease with which knowledge can be integrated incrementally into the existing store of knowledge.

Another problem with many rule-based expert systems is that one of the primary reasons for the rule representation — that is, being able to use and examine such knowledge in different ways in different situations — is rarely realised. For example, many rule-based systems are restricted to either invoking rules on the basis of facts becoming known *(data-driven*

invocation), or invoking them on the basis of certain goals becoming important *(goal-directed invocation).* Furthermore, the facts and goals that give rise to the invocation are required to occur explicitly within the rule itself, i.e., as rule antecedents or consequents, respectively. While this simplifies the representation, it is a very restrictive assumption and does not readily permit the kind of hypothetico-deductive reasoning used by most problem-solving experts [Clancey 1981].

For example, the system should be able to be goal-directed, yet also be able to alter its line of reasoning and set up different hypotheses (goals) if a particularly unusual fact is observed. One might also want a certain rule to be invoked only when a certain goal was (or was not) being attempted, yet still want it to be data driven (in the context of this goal) if a particularly significant fact was observed. For example, a lump in some tissue should perhaps invoke a rule to investigate the possibility of cancer, but not when the current hypothesis is a viral infection (where such lumps are usually caused by reacting glands).

In this paper we describe a scheme for *explicitly* representing procedural knowledge while still retaining the benefits of rule-based systems. Th" basis of the scheme is to use a representation that is sufficiently rich to describe arbitrary sequences of actions in a simple and natural way, while at the same time avoiding explicit procedure "calling" (see [Georgeff 1982]). The scheme also allows a wider class of invocation criteria than is available in most rule-based, expert systems. We shall call systems that are based on this scheme *procedural expert systems.*

To evaluate the scheme, we have implemented a system called Peritus,* a knowledge engineering tool which allows the creation of procedural expert systems specific to any chosen problem domain [Bonollo and Georgeff 1983]. Although we shall not discuss this system in any detail, for most of our examples we shall use an actual run from the domain of automobile engine fault diagnosis.

§2 The Knowledge Representation

The basic structure of a procedural expert system (PES) is similar to that of most rule-based expert systems. That is, it consists of (1) a knowledge base for storing information about both the problem domain and the specific problem being examined, and (2) an inference mechanism for manipulating this knowledge [Buchanan and Duda 1932].

The knowledge base itself comprises a data base containing *facts* about the problem and a set of specialized inference procedures called *knowledge areas* (KA).

A knowledge area consists of an *invocation part* and a *body.* The invocation part is an arbitrary logical expression that may include conditions on both currently known facts and currently active goals. A KA can only be invoked if this expression evaluates to "true", in which case the KA is considered to be potentially useful in solving the problem at hand.

The body of a KA can be viewed as a specialized inference procedure. In essence, it is simply a procedure that establishes sequences of subgoals to be achieved (facts to be discovered) and draws conclusions (establishes other facts) on the basis of achieving (or not achieving) these subgoals.

Despite the procedural nature of the KAs, the system is deductive in the sense that once a fact has been added to the data base, it cannot be subsequently deleted. Judgmental reasoning is accomplished by associating probabilities or certainties with both the facts and the inference procedures, in much the same way as for the more usual rule-based systems.

2.1 Representing Procedural Knowledge

Procedural control is specified by using a recursive transition network (RTN). The arcs of the RTN are labeled with predicates (tests) and functions (actions), in much the same way as for an augmented transition network (ATN) [Woods 1970].*

A given arc of the network can be traversed only if the predicate labeling that arc evaluates to "true". All possible paths in the RTN are explored, beginning at a specified start state and ending at a specified final state. (Of course, only those paths with arcs whose associated predicate evaluates to "true" may be followed.) This is unlike the procedure adopted for ATNs, which exit as soon as one path has been traversed to the final state. The order in which the paths are explored should be considered as undeGned (i.e., the validity of the inference procedure should not depend on this ordering). However, in the implementation described in this paper, paths are explored in a depth-first manner.

A typical KA body is shown in Figure 1. *"Start"* is the start state and *"end"* the final state of the network; the tests on arcs are indicated by a "?". Now if, for example, all the arc tests evaluated to "true", the transitions tr_1, tr_2, tr_2, tr_4 and tr_5 would all be made, and the actions a_2, a_3, a_4 and a_5 all executed, in that order, before the KA exited. If, on the other hand, test t_2 fails, then only the transitions tr_1 and tr_5 would be effected, and only action a_5 would be executed.

The functions and predicates labeling the arcs of the RTN can be any computable functions or predicates. In the Peritus system, these are specified in LISP and can make use of variables local to the KA. Free variables are not allowed.

There is also a special class of functions and predicates that access the data base and add new facts to it. Some of these predicates ask whether certain facts are true or not, and will set up subgoals to ascertain these facts if they are not currently known (i.e., not currently in the data base). We shall call these predicates *goal-invoking* predicates. As in most goal-directed systems, goal-invoking predicates can be viewed as a form of implicit subroutine call. Similarly, some of the functions labeling the arcs may draw conclusions that add facts to the data base, thus making it possible for previously requested subgoals to be achieved.

In essence, the RTN is used to define a language over the predicates and functions that label the arcs of the network (see [Georgeff 1982]). This language determines the allowable sequences of predicate and function evaluation, but not necessarily deterministically. Instead of an RTN, it may be preferable to use some other [sufficiently expressive] means of describing sequences (such as, for example, temporal logic).

* "Peritus" is the Latin word meaning "skillful" or "expert".

For example, consider the following procedure for isolating an electrical-system fault in an automobile engine that will not start [Gregory 1980]:

spark-plug test: Disconnect a spark plug lead and see if a spark jumps to the cylinder head on attempted engine start. If the spark is satisfactory or blue, then the spark plug may be faulty. If the spark is absent, weak, or yellow, proceed to the next test.

coil-lead test: (Instructions on how to test the spark from the coil) If the spark is satisfactory or blue, proceed to the next test. If the spark is absent, weak, or yellow, then the low-tension circuit is suspect; proceed to the low-tension test.

distributor test: If there is evidence to suggest that the high-tension leads from the distributor are not operating properly, conclude that they may be faulty. If the high-tension leads appear to be in order, conclude that the distributor may be at fault.

low-tension test: (Instructions about how to perform this test) If a test lamp lights on the ignition side of the coil but not on the distributor side, conclude that the coil and/or the coil lead may be faulty. If the test lamp doesn't light on the ignition side of the coil, conclude that the low-tension circuit may be faulty.

A KA corresponding to this procedure is shown in diagrammatic form in Figure 2. There are a number of things to note about the body of this KA (we shall consider the invocation part later). First., when facts are required to be established (e.g., as in "spark is satisfactory or blue") other KAs may be invoked to ascertain this information. In this case, an "ask-user" KA might be invoked, but in general many KAs might respond before the fact is established (as might be the case, for instance, in checking the status of the high-tension leads).

Second, note how very important is the order in which the facts are ascertained. The conclusions to be made at any stage are *context-de.pendent,* i.e., they depend critically on the knowledge that certain alternatives have already been ruled out (or in). In general, the conclusions may also depend on the tests and actions being carried out in a particular time order (such as listening for a scraping noise *after* applying the brakes in order to establish the condition of the disc pads). Note also just how much heuristic information is contained in this ordering, how we start off with the spark plugs rather than the coil (partly because they are easy to rectify, partly because they are the more common fault) and then proceed from one end of the circuit to the other. To discard this information (or to represent it implicitly), leads not only to inefficient problem-solving, but also to a behaviour that is difficult to explain and difficult to follow (Clancey 1981].

The RTN formalism can also represent other control constructs, including iteration and recursion. Such control constructs are needed when it is desired to examine different instances of a given object type. For example, we might have a KA (or KA "schema") for determining whether or not a single spark plug is faulty. This KA would be invoked whenever a goal was set up to determine the status of some *particular* spark plug. However, by using iteration (represented as a loop in an RTN), we can also set up this goal for any arbitrary number of spark plugs in the engine, thus creating multiple instances of the spark plug KA. The ability to explicitly establish goals in this way can be very useful when one wishes to examine different instances of a given object type in a specified order (e.g., such as testing the spark plugs in firing order) or when a certain conclusion depends on interactions among instances.

We can thus view the RTN as specifying sequences of subgoals to be achieved and, depending on the conditions obtaining, as drawing conclusions about the current problem. This is similar to the way in which, for example, rules in goal-directed systems (e.g., EMYCIN [van Melle 1980]) are used to set up subgoals to be achieved, and then make conclusions on the basis of these subgoals having been achieved. The scheme described in this paper differs in that the achievement of subgoals may be explicitly sequenced in quite complex ways, and the conclusions reached may be valid only if this sequencing is maintained.

2.2 Representing Utilitarian Knowledge

The invocation part of a KA is any arbitrary expression that evaluates to "true" or "false". A KA can be invoked only if this expression evaluates to "true", in which case the KA is considered to be useful in solving the current problem. One can thus view invocation expressions as domain-specific meta level rules [Davis 1980] that constrain the application of KAs to situations in which they are likely to be useful.*

An invocation expression can include functions that examine the current goals and functions that respond to new data. The current system uses a *goal stack* to keep track of all current goals and uses a *goal function* to test whether a given goal (or generalization) is present on the goal stack. The goals on the goal stack form an inference chain whereby for each goal g_i, the goal , (which is closer to the "top") may be viewed as a subgoal to be determined as part of the process of establishing 0i- A *fact function* is also provided to ascertain whether or not a given fact occurs in the data base.**

An invocation expression consisting solely of the goal function results in standard goal-directed invocation, whereas an expression consisting solely of the fact function results in standard data-driven invocation. For example, two simple KAs from the automotive domain are shown in Figure 3. In Figure 3(a), the KA is goal-directed, and corresponds to a standard

*As in MYCIN-like systems [Shortliffe 1976], a request for a fact establishes a goal that is a generalization of the fact. Thus, any subsequent test for an instance of that generalization will find the necessary information already in the data base and will not need to reinvoke KAs to determine it.

* Invocation expressions serve solely to determine the set of useful KAs, not to order them. In this sense they are a more limited form of metarule than discussed by Davis. On the other hand, they are more general in that they can "rule in" some alternatives that under a purely goal driven or data driven system would not be considered.

** Thc implemented system uses judgmental criteria and a minimum certainty factor can be required of the requested fact. This allows KAs to be invoked on the basis of incomplete or uncertain knowledge.

MYC IN-like rule. This KA will be invoked only if *some* current goal is to identify a fuel system problem, and the required information is not currently known (i.e., in the data base). Invocation of the KA will then test whether there is reduced fuel flow in the fuel pump, possibly invoking other KAs to ascertain this. If reduced fuel flow is concluded, the [single] arc of the network can bo traversed and the fuel system problem identified (with some degree of certainty) as being a choked suction line.

An example of a data-driven KA is given in Figure 3(b). Note that the test in the body of the KA seems redundant in this case, as the KA would not have been invoked if it had not already been known that the oil was contaminated with water. However, it is important from a methodological point of view to require that the body of a KA be valid irrespective of the invocation condition. Under these conditions, if the inference procedures forming the bodies of all the KAs in the knowledge base are consistent, the system will be consistent. Alteration of the invocation conditions to improve performance will not affect the validity of any conclusions made, thus allowing the system to be "tuned" in safety. (Of course, completeness is not guaranteed; because of inappropriate invocation criteria a certain KA may never be invoked and thus a valid inference never made.) The apparent inefficiency of having to check the data base twice for (in this case) the condition of the oil could be eliminated during a compilation stage.

KAs can also be partly goal-directed and partly data-driven, as is the case for the KA shown in Figure 2. It will be invoked if one of the current goals is to identify an ignition fault and it is noticed that one of the trouble symptoms is that the engiue does not start. The system can thus be opportunistic in the sense that KAs might be invoked because certain facts are noticed during an attempt to establish particular goals.

In general, any arbitrary expression can be used in the invocation part of a KA. Such expressions can include negation, conjunction, and disjunction of both goals and facts. Universal and existential quantification over *existing* object instances are also allowed.

§3 The Inference Mechanism

The system's main task, at a particular point in time, is to discover all it can about the current goals by executing relevant KAs. To do this, an invocation mechanism is called implicitly by the currently executing KA when some currently unknown fact is requested or when some new conclusion is drawn. The mechanism evaluates the invocation part of all *instances* of the KAs occurring in the knowledge base to decide which ones are "relevant" (i.e., for which of them the invocation part evaluates to "true"). These relevant KAs are then executed or invoked in turn until either they have all been executed or a definite conclusion has been reached about one of the current goals on the goal stack.

The invocation mechanism is outlined in Figure 4. The set *S* is initialized to contain all relevant instances of the KAs occurring in the knowledge base. The function *eclect(S)* [destructively] selects an element *p* from the set of applicable instances of KAs 5, and e*xecute(p)* executes the body of *p*.

Of course, for efficiency purposes (and to prevent asking the same question more than once) we do not want to reinvoke instances of KAs that have already been traversed. We therefore mark these instances as "used", which effectively deletes them from the knowledge base.

Execution of a KA body consists simply of traversing the body of the KA, as described in Section 2.1. In fact, Peritus actually compiles the networks into LISP code, in a manner similar to ATN compilers (see [Bonollo and Georgeff 1983]). This makes the system much more efficient than if KAs were evaluated interpretively.

It is very important to note that although the body of a KA sets up sequences of goals to be achieved, it may be that during its execution some data invoked KA suggests an alternative hypothesis and thus changes the course of events. Progress through the original KA is then suspended, and will only be resumed when the alternative hypothesis has been fully investigated. If we had more control over the selection of tasks (say, by using metalevel KAs), the currently executing KA could also be suspended simply because other goals (hypotheses) became *more* interesting. Thus, it is preferable to view KA bodies as placing *constraints* on the sequencing of goals, while not precluding the possibility that certain observations may (at least temporarily) interrupt this sequencing.

3.1 Definite Conclusions about Goals

Elements on the goal stack represent hypotheses that the system is trying to establish. Once a hypothesis that occurs on the goal stack is confirmed *with certainty,* it is pointless to continue considering alternative paths in the current KA. Furthermore, if this hypothesis is not on top of the goal stack, it is also desirable to terminate all KAs that are trying to establish additional supporting hypotheses (i.e., all subgoals of the just-concluded goal).

The current implementation achieves this by "tagging" calls to KAs with the goal that was the top goal at the time a particular instance of the invocation mechanism was called. A conclusion as to this goal can then cause immediate exit of the current KA being explored. This results in an explicit alteration of the normal function return sequence by "throwing" * to the instance of the invocation mechanism whose "tag" is the hypothesis that has just been confirmed. Any intermediate calls to this mechanism are discarded and the goal stack is automatically restored to a state in which the just-concluded goal is the top goal.

However, this can present difficulties when it comes to deciding which instances of KAs should be marked as "used" and hence not available for reinvocation. The problem is that a "throw" to some previous state in the goal stack can pre-empt complete traversal of some KAs (i.e those that were working on subgoals of the just-concluded goal), and we might be throwing away the knowledge needed to reach other conclusions (about other goals) if these KAs were not able to be reinvoked.

The solution currently adopted is to mark these partly used KAs as fully used, thus possibly jeopardizing the completeness of the system but not its consistency. This is not entirely

*This is implemented via the catch and throw primitives present in most LISP implementations (e.g., FRANZ LISP [Foderaro 1980]).

satisfactory, however, and other alternatives need to be investigated. Tor example, it might be better if partly used KAs were left unmarked (i.e., unused), allowing for possible reinvocation. It would then be important to ensure that multiple re-use of an instance of a KA would not affect the validity of any conclusions made, and that i/o operations were encased in KAs that always exited naturally (i.e without "throwing").

3.2 Incorporation of Judgmental Knowledge

The scheme described above can readily be extended to include judgmental mechanisms for application in domains where facts and inference rules (or inference procedures) exhibit some degree of uncertainty. In the Peritus system, the judgmental scheme is based on certainty theory [Shortliffc and Buchanan 1975]. However, procedural expert systems are not tied irrevocably to this philosophy and there is no reason not to apply other criteria in dealing with judgmental knowledge.

To implement, the judgmental mechanism, each arc test in the body of a KA is enclosed in an interface function. The purpose of this interface function is to provide judgmental criteria for deciding whether the result of a test is "true enough" to continue progress along the are. In the current system, the criteria used are as follows :

i. Any test that returns nil is mapped to a certainty factor of 0.

ii. Any test returning a non nil value that is not a certainty factor is mapped to a certainty factor of 1.

iii. Any test that returns a certainty factor is left unchanged.

The interface function then returns "true" if this new certainty factor is greater than an arbitrary threshold of 0.2.

As the functions which add new facts to the data base need to know the certainties associated with each of the tests (or premises) that precede them, a stack of the certainties associated with the tests on the current path is maintained and can be accessed by these conclude functions. When an arc test evaluates to "true", the certainty factor associated with that test is pushed onto the stack as a side effect of the interface function. The stack is popped at the end of a path or after the last transition from a state has been attempted.

3.3 A Sample Consultation

Portions of a sample consultation in the domain of automobile engine fault diagnosis are given opposite. The procedural expert system that produced this dialogue was constructed by using Peritus, the pertinent knowledge being extracted more or less directly from various automobile literature (e.g., [Sully and Unstead 1978], [Gregory 1980]). Typical examples of the KAs used by the system are shown in Figures 2 and 3.

[Various initial data is first entered]

>Does the engine start? *no*

[At this point, a data driven KA is invoked on the basis of the engine not starting]

FUEL CHECK!!
>Is there adequate fuel in the tank? *yes*

[and now the KA shown in Figure 2. is invoked]

Ignition System Trouble Shoot Procedure:
(1) Disconnect a spark plug lead and position the terminal within 3 mm of the cylinder head.
(2) Have an assistant switch on the ignition and attempt to start the engine
(3) Observe the spark, if any, that jumps the gap to the cylinder head
Press the <return> key when you are ready —»

>Ilow would you describe the spark? *why?*
[The user wants to know why the system needs this information]

I was trying to determine one of the ignition system faults, and I noticed that the engine does not start.

[The user now continues with the trouble shoot]
> *weak*

(4) Disconnect the coil lead from the distributor cap and carry out the procedure previously described for the spark plug lead in points (1) to (3).
Press the <return> key when you are ready —►

>Ilow would you describe the spark from the coil lead? *weak*

(5) We will now check whether the low tension current is reaching the ignition coil. Use a test lamp between the coil low tension terminals and a good earth. Ensure that the contact breaker points are open or insulated when making the test.

Press the <return> key when you are ready —►

>Does the lamp light on the ignition side of the coil? *yes*
>Does the lamp light on the distributor side? *no*

[The system has now discovered all it can and gives its report]

Automobile Suspected Faults Report:
There is evidence (0.8) to suggest that the ignition coil is faulty.
There is evidence (0.5) to suggest that the coil lead is faulty.

§1 Conclusions

Procedural expert systems appear to provide a means for representing a specialist's declarative and procedural knowledge of some problem domain without sacrificing any of the desirable properties of standard rule-based expert systems.

There are a number of important features of the system that are critical to achieving this. First, KAs are not directly "called", but are invoked only when they can contribute to finding some current goal or when some particularly relevant fact is observed. As KAs cannot be directly called, neither can they directly call any other KA. They thus serve only to specify what goals are to be achieved and in what order. Second, the system is, in general, nondeterministic, and any number of KAs may be relevant at any one time. These properties enable the knowledge base of the system to be modified and augmented without forcing substantial revision.

Furthermore, the representation of the inference procedures in the form of an augmented RTN is simple and homogeneous, which aids both in the acquisition of knowledge and in verifying correctness. This simplicity also aids the system in explaining its reasoning. In the simplest case, the goal stack can be traversed to answer "how'' and "why" questions (as in MYCIN-like systems). Hut, by tracing the bodies of the KAs as well, the system can also describe the context in which hypotheses are being explored. This kind of explanation is of course not deep, but the formalism itself does not preclude the development of a richer explanation system. Indeed, the fact that at least some of the procedural knowledge of the expert is explicitly represented can only lead to better explanatory capabilities.

The procedural control component is also very general, allowing at one extreme the construction of purely declarative programs, while, at the other extreme, it allows purely deterministic procedural programs. Thus, procedural expert systems bridge the gap between the declarative languages and the procedural languages. Indeed, one way of viewing these systems is as a generalization of the usual procedural languages to include a deductive data base and a more general invocation mechanism.

The fact that the knowledge representation allows the specification of procedures means that the inference mechanism of the system can itself be written by using the same representation. For example, the description of the current invocation mechanism given in Figure 4 is already in this form.

The generality of the invocation scheme makes it possible for the system to pursue a diagnosis in a goal-directed way, yet react opportunistically and change the direction of the consultation if an event occurs that suggests an alternative diagnosis. In fact, after using primarily goal-directed systems like MYCIN, the way in which data invoked KAs suddenly wake up and begin exploring alternative diagnoses was a constant surprise to the authors (not always pleasant!). As invocation expressions can be arbitrarily complex logical expressions, they can represent quite complex knowledge about the use of KAs. For example, although we have not explored it here, the scheme allows invocation based on observed differences between goals and facts, and KAs can thereby be invoked in a means-ends fashion.

There are a number of questions that still remain to be answered, and this will require further experimentation with the system. For example, it may be that the class of invocation expressions used by the present system is too broad. When arbitrary expressions are allowed, knowledge about the use of KAs can be very difficult to reason about. This might create problems if more detailed explanations were required or if metalevcl KAs were to try to manage invocation. Furthermore, there would then be fewer possibilities for compiler optimizations, which could result in unacceptable real-time performance.

References

[1] Bonollo, U. and Georgeff, M.P. "Peritus: A System that Aids the Construction of Procedural Expert Systems", to appear as Computer Science Tech. Report, Monash University, Melbourne, Australia, 1983.

[2] Buchanan, B.C., and Duda, R.O., "Principles of Rule-Based Expert Systems", to appear in M. Yovits (ed.) Advances in Computers, Vol 22, Academic Press, New York.

[3] Clancey, W.J., "The Epistemology of a Rule-Based Expert System: A Framework for Explanation", to appear in Artificial Intelligence.

[4] Davis, R., "Metarules: Reasoning about Control", Memo AIM 576, MIT AI Lab, MIT. Mass., March 1980.

[5] Foderaro, John K., The FRANZ LISP Manual, University of California at Berkeley. 1980.

[6] Georgeff, Michael P., "Procedural Control in Production Systems", Artificial Intelligence, Vol 18, pp. 175-201, 1982.

[7] Gregory's SP Manual No. 165: Falcon Fairmont XD Series 6 Cylinder Service and Repair Manual, Gregory's Pub. Co., Sydney, 1980.

[8] Michie, D., "Expert Systems", The Computer Journal, Vol 23, pp 369-376, 1980.

[9] Reboh, R. "Knowledge Engineering Techniques and Tools in the Prospector Environment", Tech Note 243, Artificial Intelligence Center, SRI International, Menlo Park, Ca., 1981.

[10] Shortliffe, E.II. Computer Based Medieal Consultations: MYCIN, American Elsevier, New York, 1976.

[11] Shortliffe, E.H. and Buchanan, B.C., "A Model of Inexact Reasoning in Medicine", Mathematical Biosciences, Vol 23, pp 351-379, 1975.

[12] Sully, F. K. and P. J. Unatead., Automobile Engines Questions and Answers, 3rd. Ed., Newnes Technical Books, Buttcrworth and Co. (Publishers) Ltd, London, 1978.

[13] van Melle, W. "A Domain-Independent System That Aids in Constructing Knowledge-Based Consultation Programs" Memo HPP-80-1, Report No. STAN-CS-80-814, Computer Science Department, Stanford University. June 1980.

[14] Winograd, T., "Frame Representations and the Declarative Procedural Controversy", in Bobrow, D. and Collins, A. (Eds.). Representation and Understanding , Academic Press, New York, 1975.

[15] Woods, W. A., "Transition Network Grammars for Natural Language Analysis'*, Comm. ACM., Vol 13, 1970.

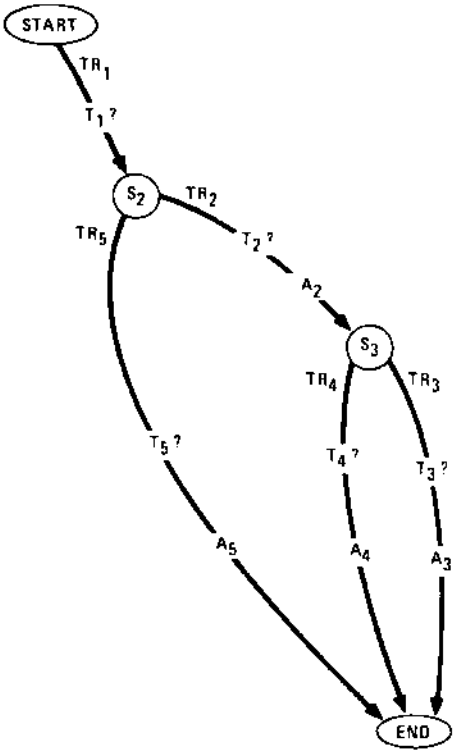

FIGURE 1 A TYPICAL KA BODY

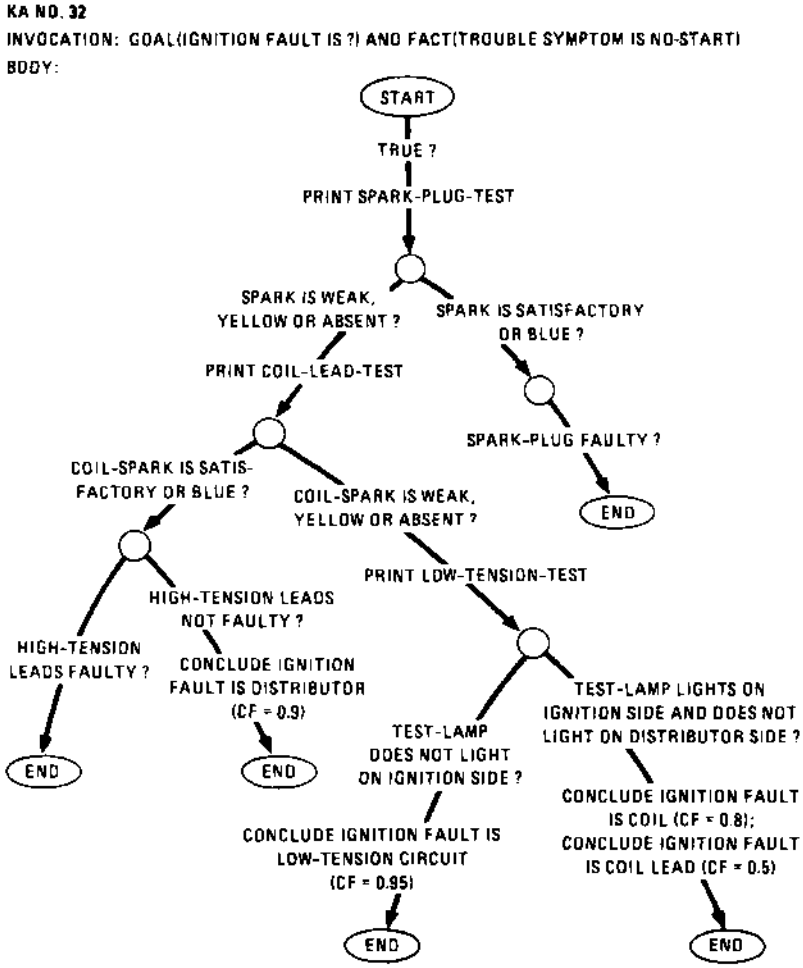

FIGURE 2 A SAMPLE KA FROM THE AUTOMOTIVE DOMAIN

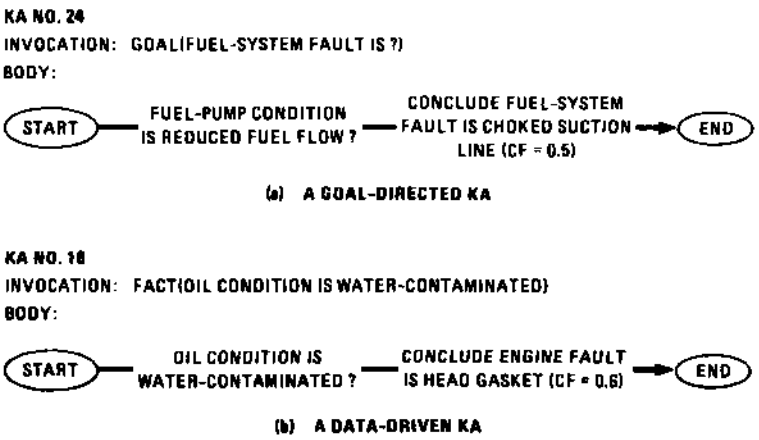

FIGURE 3 SOME SIMPLE KAs FROM THE AUTOMOTIVE DOMAIN

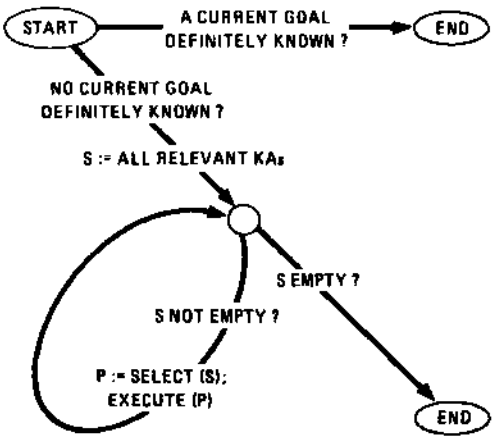

FIGURE 4 THE INVOCATION MECHANISM