which additional steps should be pursued, even though they eventually prove to be false leads, and (3) which steps do not make any sense. It also knows at any point during the conversation where the student stands, i.e., what factual information has been obtained, what conclusions can be drawn from these facts, and what additional information is needed to draw further conclusions. MEDCAT uses these two perspectives to direct its questioning of the student.

MEDCAT exhibits a tolerance and understanding that befits its role as a teacher. The primary reason for this is that it focuses on the logic of the student's reasoning, rather than the absolute aspects of right or wrong answers that limit so many computer teaching systems. There are many routes to a correct diagnosis, and even paths that ultimately prove to be blind alleys may be worth pursuing. Indeed, these may require investigation. The program can accompany the student along these paths and respond to his reasons, while patiently waiting for a point at which the need to look elsewhere becomes obvious. The fact that the program has a more comprehensive perspective does not mean that this must be forced upon its pupil prematurely. The program exhibits intolerance only in the realm of observations (i.e., empiric data). When the student fails to obtain essential information, forgets what he has already been told, or—most of all—assumes signs or symptoms or laboratory data that have not been elicited by the questions, the program becomes quite critical.

CATS. Compared to MEDCAT, CATS is relatively intolerant, because CATS deals mainly with empiric information. In the data-acquisition phase of MEDCAT, the HBSAG is either positive or negative, or was not done. Similarly, in CATS, a muscle either is or is not innervated by the musculocutaneous nerve; there is little room for debate. Where variations do occur, as in the section on anatomical variations, the program discusses them, but such situations are rare.

Instead of following alternative paths of reasoning, CATS focuses on what the student should know and organizes this in the most efficient sequence possible. When the student is wrong, CATS not only corrects him, but also points out general principles that make the correct answer easier to remember, as discussed in the section on implementation of logic.

The basic strategy in CATS is to quiz the students until they demonstrate a predetermined level of competence. The reason for having a student mode, in which the program asks the questions, is that students prefer spending their time being quizzed, rather than asking questions of their own. The program is sufficiently detailed that we can tell the students that they will never be asked anything on the written exams that the program cannot answer. If they operate in student mode for a brief time, they will be asked every type of question we can ask. If they complete the entire quiz, they will have seen every question we can ask them. For each of the five major regions of the body (e.g., upper limb), this requires a minimum of 20 hours.

The program records every question a student has been asked as well as the correctness of each answer.

The subject matter is temporally organized so the students are asked only about material they have already seen in the laboratory. Within this framework, the topics are hierarchically arranged so that the program picks a muscle group (in the extremities), and then asks about each of the muscles in that group. The program uses this as the contextual basis for asking about the innervation, blood supply, attachments, relations, functions, etc. of that particular structure. One reason for focusing on groups—indeed, the basis for arranging muscles into groups—is that it emphasizes the redundancy. There is a lot of overlap between muscles within such a group as to their nerve and blood supply, origins, insertions, and actions.

The program records every question a student has been asked as well as the correctness of each answer. If the answers about the details of the nerve innervating one muscle in the group are correct, for example, those questions are omitted for similar muscles. Students are permitted to choose the level of proficiency they want to attain. If a student picks 75 percent proficiency, for example, and more than 25 percent of the answers to questions about a muscle are wrong, that muscle is asked about again at some point.

After the program has completed a muscle segment, the students are permitted to ask questions of their own. In this way, a student may ask for general principles ("Why?") or ask to clear up misunderstandings that may have arisen.

We have stated that gross anatomy is a descriptive science; it also is a visual discipline. Students are expected to be able to draw the structures that comprise the human body. For this reason, the answers that CATS gives are accompanied by detailed anatomical illustrations generated on the screen. These drawings are displayed via the Enhanced Graphics Adaptor (IBM PC/AT).

Meaning and the representation of thought

One of the more elusive problems in artificial intelligence research is how to represent an idea abstractly, i.e., independently of the exact words used to express the thought. We need to be able to do that if we expect to simulate human reasoning. In a sense, that is what this paper is about. In order to bring this into clearer focus, we briefly summarize some of the features that directly address this question.

The internal representation. The term *semantic network* did not originate with us. "Semantic networks are a very popular representation scheme in artificial intelligence. Node-and-link structure captures something essential about symbols and pointers in symbolic computation and about association in the psychology of memory."[6] However, within this general framework, the exact methods of implementation show much variability. MYCIN (another medical diagnostic program) does not use a semantic network; it is a production-rule-based system.[7]

Because of our neurophysiologic background,[8] our particular model is based on the microarchitecture of the brain. Briefly stated, a neuron (node) discharges if the algebraic sum of the effect of all its afferent axons (pointers) exceeds its threshold. When it fires, it does so in an all-or-nothing fashion. The efferent fiber (pointer) from this neuron (node) influences (facilitates or inhibits) the firing of other cells (nodes).

Because we use APL, the neuronal net is numeric. This pointer matrix represents both the subject matter (names of things) and the logic (relations among these concepts). The pointers may indicate such diverse things as the strength of a relationship and the nature of the association, and reasons for or against decisions made by the system. Not only may this matrix act as the mechanism to perform the reasoning, but these same numbers may also be translated directly into the noun and verb phrases that they represent to describe the reasoning. This makes for a model that is very easy to work with for all concerned—the programmer, the subject matter expert, and the user.

Contextual understanding on a coding level is the focus of the research reported in this paper.

Such a numeric representation is not the norm in the artificial intelligence field. One textbook states: "Artificial intelligence without one of these (Lisp-like) languages is physics for poets—laudable and useful, but not completely serious."[9] Those who use list-processing languages describe difficulties and concerns that we have not experienced, including "computational problems that arise when network databases become large enough to represent nontrivial amounts of knowledge."[6] APL, on the other hand, is designed to deal with numbers and is replete with primitive functions to traverse such arrays. The pointer matrix used involves direct relations between pairs of nodes. Therefore, we do not become involved with search strategies or problems arising from the depth of search.

The contextual basis of meaning. The meaning of an individual word depends on the context in which it is used. On a behavioral level, this is obvious, but it has never been implemented on a programming level.[10] For example, lack of this ability has been a major impediment in the development of machine translation of foreign languages.

Quillian[11] approached the problem by emphasizing the role of associative links between individual words in the representation of meaning. His research began at a more basic level, which included using pointers between words to define the words themselves. Com-

binations of these words, once defined, were then used to express concepts.

Contextual understanding on a coding level is the focus of the research reported in this paper. Because it may be the most significant contribution, we now restate the essential elements to illustrate this feature.

Our model does not contain any pointers between individual words. It begins with a naturally occurring collection of concepts (the names of things) expressed in the form of noun phrases. The pointers run from the word to all those phrases in which it may be found. The fact that these node descriptors are augmented with synonyms means that these nodes may be identified and recognized by whatever diversity of phrasing our language permits.

On an objective level, the resulting discussions sound more natural than other computer-mediated dialogues dealing with similar domains.

Noun phrases have evolved to eliminate ambiguity. It is their function to uniquely establish the simplest level of context. Ambiguity reappears in the presence of noise (typographical errors) in the input, or when the user omits some of the words because, on a human level of understanding, the context is established by other means. The verb commonly provides these more subtle semantic clues. The program can interpret this meaning by using the expected profile. Additional methods of making the context more specific are demonstrated by predicate adverbials and prepositional phrases. This is as far as this program has reason to go. However, just as the meaning of a word may be clearer in a sentence than it is in an isolated phrase, so other words in a paragraph (outside the sentence) may modify the meaning within that sentence.

The natural-language interface. Lexical information (word typing) is stored only for the syntactic markers (i.e., conjunctions, determiners, helping verbs, prepositions, punctuation) that serve to break the input into functionally useful units. Information-containing words (i.e., adjectives, nouns, verbs) are mapped directly onto the pointer matrix where they will be used, after the ambiguity resulting from their overlap has been resolved by the contextual clues already described. On a speculative level, this is much closer to the way people process language than traditional linguistic theory would suggest. On an objective level, the resulting discussions sound more natural than other computer-mediated dialogues dealing with similar domains.

Concluding remarks

We have described in some detail three aspects of our two programs CATS and MEDCAT—the data structure, the logic based on these structures that may be used for answering questions, and a natural-language system for making these features available to the user. The simplicity of each of the individual components and the way in which they interact suggest that others may want to apply similar techniques to different problems. The fact that the two programs—with very similar code and data structures—can handle two such diverse subjects strongly implies that this general approach should find other educational uses, wherever the ability to reason plays an important role. APL happens to be particularly well suited for this type of application and these types of data, but implementation certainly is not language-dependent.

Acknowledgments

This work was supported in part by grants from the Anonymous Gift Fund of Cornell University Medical College, the Frances L. and Edwin L. Cummings Memorial Fund, and the Advanced Education Projects of the IBM Corporation. We would like to thank Mr. Ward Bell for writing certain auxiliary processors used in the 8086 version of the programs and Dr. Grace Hucko for her helpful suggestions.

Cited references

1. W. D. Hagamen, M. Gardy, G. Bell, E. Rekosh, and S. Zatz, "MEDCAT: An interactive computer program for medical diagnosis," *Proceedings of NCC'85*, Chicago, IL, July 1985, Vol. **54**, pp. 111–119 (published by AFIPS Press, Reston, VA).
2. W. D. Hagamen and M. Gardy, "MEDCAT/CATS: Two contrasting artificial intelligence applications in medical education," *Proceedings of the Second Conference on Artificial Intelligence Applications*, Miami Beach, FL, December 1985, pp. 503–508 (published by IEEE, Los Angeles, CA).

3. W. D. Hagamen, D. J. Linden, K. F. Mai, S. M. Newell, and J. C. Weber, "A program generator," *IBM Systems Journal* **14,** No. 2, 102–133 (1975).
4. D. D. McDonald, "Surface generation for a variety of applications," *Proceedings of NCC'85*, Chicago, IL, July 1985, Vol. **54,** 105–110 (published by AFIPS Press, Reston, VA).
5. W. D. Hagamen, D. Linden, M. Leppo, W. Bell, and J. C. Weber, "ATS in exposition," *Computers in Biology and Medicine* **3,** No. 3, 205–226 (1973).
6. A. Barr and E. A. Feigenbaum, *The Handbook of Artificial Intelligence I*, William Kaufman, Inc., Los Altos, CA (1981), p. 189.
7. B. G. Buchanan and E. H. Shortliffe, *Rule-Based Expert Systems*, Addison-Wesley Publishing Co., Reading, MA (1984).
8. W. D. Hagamen, *The Functioning Brain of Man*, Chas. Pfizer & Co., Inc., New York, NY (1966).
9. P. H. Winston, *Artificial Intelligence*, Addison-Wesley Publishing Co., Reading, MA (1977), p. 263.
10. T. Winograd, "Computer software for working with language," *Scientific American* **251,** No. 3, 131–145 (1984).
11. M. R. Quillian, "Semantic memory," in *Semantic Information Processing*, MIT Press, Cambridge, MA (1968), pp. 216–270.

Wilbur D. Hagamen *Professor of Cell Biology and Anatomy, Cornell University Medical College, 1300 York Avenue, New York, New York 10021.* Dr. Hagamen received his M.D. from Cornell in 1951, and has taught neurosciences and/or gross anatomy there since 1949. Until 1970, he did research in neurophysiology. Dr. Hagamen spent a sabbatical year (1972–73) at the IBM Systems Research Institute. From 1972–79 he was Director of the Laboratory of Computer Science at CUMC and implemented all the data processing needs of CUMC/NYH in APL (fiscal, patient records, and research). Dr. Hagamen also wrote APL programs for various financial institutions, including the entire Standard & Poor's Blue List Retrieval System, which ran on the CUMC computer. His computer research interests focus on artificial intelligence, natural language, and interactive graphics.

Martin Gardy *Associate Professor and Associate Chairman of the Department of Medicine, Cornell University Medical College, 1300 York Avenue, New York, New York 10021.* Dr. Gardy is in charge of the Department's teaching program. He is also the Clinical Coordinator for the Rockefeller University/CUMC M.D.-Ph.D. program. Dr. Gardy is a graduate of the City College of New York, where he received his B.S. in 1956, and CUMC, where he received his M.D. in 1960. He has received many teaching awards, including the New York State Medical Society's award for Excellence in Clinical Teaching. Dr. Gardy's major interests are diagnostic problem solving and decision analysis.

Reprint Order No. G321-5272.

The Portable Inference Engine: Fitting significant expertise into small systems

by N. A. Burns
T. J. Ashford
C. T. Iwaskiw
R. P. Starbird
R. L. Flagg

The Portable Inference Engine (PIE) is the nucleus of an expert system that allows the segmentation of rules in order to utilize large knowledge bases in limited memory. If the expert-systems rules can be divided into segments of highly related rules with little interaction among those segments, such knowledge-base segments can then be paged in and out of memory on demand. PIE gathers information by querying the user and executing external procedures in order to conclude goals.

Expert systems are traditionally used to solve problems in a limited domain, so that a few thousand rules may be written to represent the knowledge needed to solve problems in that domain. These rules generally require a large amount of computer memory, with this knowledge usually being stored in a single large knowledge base.

Recently, several expert systems have become commercially available to run on personal computers.[1] However, these systems are all limited by the amount of real memory available in the machine on which they are running. Large problems that require more rules than can be held in memory cannot be solved with these systems. This paper describes the Portable Inference Engine (PIE), an expert system that utilizes the concepts of segmentation and paging commonly used in operating systems to overcome memory limitations.

The next section describes the problems that had to be overcome in designing an expert system of this type. Then the architecture of the system designed to solve these problems is presented, followed by a description of the operation of the expert system. Finally, the contributions of this work are summarized.

Problem characterization

PIE was initially developed as a tool for controlling hardware fault diagnostics for the IBM RT Personal Computer by extending the General-Purpose System for Inferencing (GPSI). This mission involved several challenging requirements. For example, suppose that the system is running on a machine suspected of having malfunctioning components. These components may produce unreliable or incomplete information with which the expert system must deal. In order to minimize the number of hardware and software components needed to perform the diagnostics, the operating system, the entire expert system, and the rule base are required to reside on diskette and operate in one megabyte of nonpaging

memory. As a result, code and rule-base size are of primary concern. The expert-systems shell is required to be independent of the hardware and the operating system so that this tool can be used in the development of future expert systems.

A goal of the expert system was to avoid interaction with the user when possible, so that a consultation session should involve the hardware rather than the human operator. Information found in the machine itself, such as error logs and status words, is much more reliable than user input in isolating hardware problems. The results of tests performed on hardware can provide a clear indication of the problem encountered.

Since real memory size is limited and hardware configurations are variable and volatile, a method of segmenting the knowledge base is needed. When the expert system concludes that a particular component is present and needs testing, the rules necessary for the diagnostics of that component can be read in and utilized in isolating machine faults.

Architecture

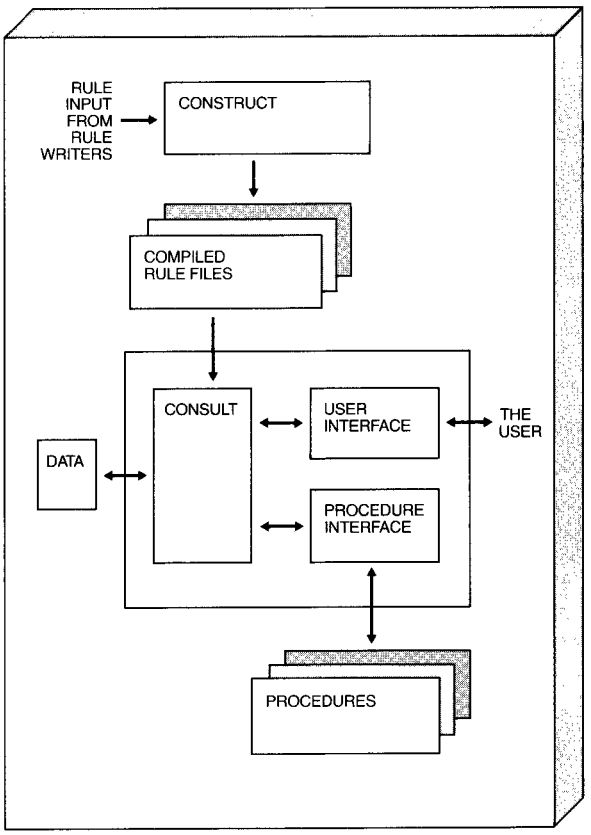
The resulting expert system for the RT Personal Computer diagnostics is structured as illustrated in Figure 1. A system checkout shell presents menus to the user and obtains information about the type of testing to be performed. It calls the inference engine, CONSULT, as a subroutine, passing it a list of values to be used in controlling the inference process. The inference engine reads in the rule base and gathers evidence by asking questions or running procedures. It then formulates goals indicating which parts, if any, are faulty, and returns the list of goals to the system checkout program.

All rules used in the expert system are precompiled by the CONSTRUCT program into two files. One file contains the information necessary for building the data structures to be used during consultation sessions. The other file contains the text for asking questions and reporting goals. In order to save memory space, records from the text file are read in one at a time when needed.

The main CONSULT module consists of a supervisor which is invoked by the system checkout shell. It communicates with the operating system throughout consultation whenever new rule-base segments must be invoked or power must be turned off.

The inference algorithm, which performs such tasks as the selection of goals, the chaining through trees,

Figure 1 Architecture of PIE



and the investigation of evidence structures, is also contained in the main CONSULT module. This inference algorithm is explained in detail in the following section.

Two independent modules that are bound along with the inference engine contain all machine-dependent code; those are the User Interface Routines and the Procedure Call Coordinator. The User Interface Routines provide the code necessary for formatting, displaying, and retrieving information from the terminal screen. For the diagnostic application, a full-screen, menu-driven interface is implemented.

The Procedure Call Coordinator for the diagnostics locates the code for a particular procedure on the diagnostic diskette and loads this code into memory, dynamically binding the code to provide addressability for the expert system when so requested. In addition, it builds input and output buffers for parameters passed between the two modules and manages the invocation of the procedure on a procedure call request from the rule base being investigated.

Since all machine-dependent code resides in the Procedure Call Coordinator and the user interface module, the porting of the system can be done easily, as has been demonstrated by porting the expert-systems shell to a machine with a different architecture in one five-day work-week.[2] Currently, there are modules which make it possible to run the system on a System/370 under VM/CMS with a glass teletype interface, and on the RT Personal Computer with a full-screen menu interface.

Each node of a rule tree has associated with it a confidence factor.

The inference process

GPSI was chosen as a basis for the expert system needed to meet these challenges.[3] GPSI was developed at the University of Illinois under the funding of the IBM Scientific Center in Palo Alto, California, and was designed for diagnostic and interpretive applications. Written in Pascal and running on an IBM Personal Computer, GPSI uses a rule base that is composed of a "forest" of trees.[4,5] Each tree contains a goal at its root and evidence needed to substantiate this goal at its leaves. This structure reflects quite well the complex, interrelated nature of hardware diagnostic rules.

In order to meet the requirements for the hardware diagnostics, the ability to invoke external procedures to perform hardware tests had to be added to GPSI. In addition, support had to be added to that system to segment the rule bases and load those rule bases in and out of memory.

The knowledge base that drives the diagnostics is represented as a forest of one or more *n*-ary trees. At its root, each tree contains a goal to be concluded or rejected. The leaves of the tree contain evidence of several different kinds that can be acquired by querying the user, executing external procedures, or referencing other nodes, trees, or subtrees.

Between the root goal and the various leaf evidence are internal nodes representing a variety of functions. AND, OR, and NOT nodes can be used to relate evidence logically. OR nodes are substantiated as soon as any node under the OR function is substantiated. AND nodes are invalidated as soon as any node under the AND function is rejected. Variations of the AND node can cause every node under the AND to be evaluated. Special control structures and sequencing operations can be introduced with the ALTERNATIVE, IF, and PREEMPT nodes.

The structure of a sample rule tree is illustrated in Figure 2. This example shows a rule intended to determine whether a printer element needs to be replaced. This rule is accepted if some initial evidence indicates that the printer is suspected of being faulty, and either a test run on the printer returns evidence that this is true, or the user indicates that the symptom of bad printing is occurring.

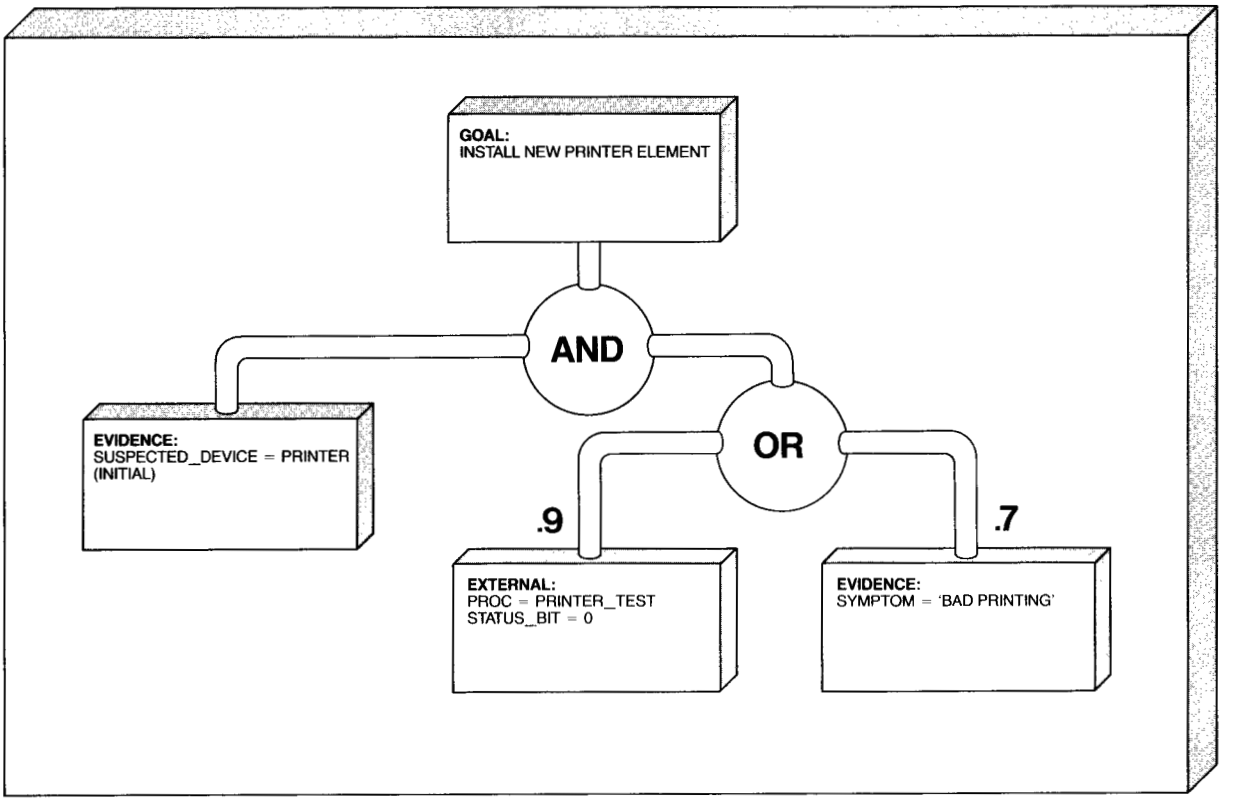
During the consultation session, conclusions are made by the inference engine primarily by using backward chaining. Once a potential goal is selected, the underlying tree is traced in a post-order traversal which prunes off unnecessary branches and gathers any necessary evidence. The type of evidence to be gathered is indicated by the type of leaf nodes on the tree. An EVIDENCE node indicates that data should be obtained by asking the user a specified question. An EXTERNAL node indicates that data will be obtained through the execution of a specified procedure. A REFERENCE node indicates that a (sub)structure in some other rule must be investigated to obtain the required information.

Each node of a rule tree has associated with it a confidence factor. For an EVIDENCE or an EXTERNAL node, this confidence value is based on an association factor given to the node by the knowledge engineer and on the answer to the question asked or the value returned from the procedure. A REFERENCE node assumes the confidence value of the structure that it references. The confidence values of other nodes are calculated from the confidence values of the "children" of the REFERENCE node. Computations used for these nodes are dependent on the type of the node.

The inference algorithm of PIE takes the following actions:

1. The confidence items for all evidence items and trees designated as INITIAL are obtained.
2. On the basis of INITIAL evidence, a tree is selected and searched to find unevaluated evidence items.

Figure 2 Sample rule tree in PIE



3. Necessary evidence is obtained and evaluated and any actions associated with true nodes are performed.
4. Other relevant nodes and structures are updated concurrently.
5. New trees are selected and fired until all trees have been traced.
6. The rule base is exhausted and all concluded goals are presented to the user.

Asking questions and calling procedures. Traditionally, expert systems have been used in consultation sessions with users. When information has been gathered from question and answer sessions, the inferencing engine makes conclusions or recommendations and presents them to the user. PIE can be used in this manner. A section of the rules, CLASSES, is devoted to defining questions and valid responses to these questions. Text for questions may be specified, and more detailed explanations about complex questions given. Information about answers may also be specified in this section, such as how many responses may be given, the type of value expected, and the valid range or list of values that an answer may assume.

For an expert system to be used as a hardware diagnostic supervisor, it must be able to execute test units. A test unit is a software procedure that performs one or more tests on the hardware and returns results to the calling program. Providing an expert system with the ability to run test units requires the capability for external procedure calls.

A section of the rule base is devoted to defining external procedure calls. Each procedure definition specifies the name of the external procedure to call, the values to be passed to the procedure, and the type of values that will be returned.

Procedure calls and class questions may be invoked from the rule trees in the rules section or designated INITIAL and evaluated before the tracing of the rule trees is begun. An EVIDENCE node specifies which question to ask, and an EXTERNAL node specifies the

Figure 3 Example rule definition

```
CLASSES
  MONITOR_OK
   TEXT = 'DOES YOUR DISPLAY MATCH THE DIAGRAM ON PAGE 10?'
   VALUES = 1 OF ('YES' 'NO')

PROCEDURES
  DISPLAY_TEST
   PROC = TU4300
   PASS  1000
   RETURN RETURN_CODE INTEGER
  ENDPROC

RULES
 1 GOAL
   TEXT = 'YOUR MONITOR IS BAD.'
 2 AND
 3 EXTERNAL
   PROC = DISPLAY_TEST
   RETURN_CODE = 0
 3 EVID
   CLASS = 'YES' OF MONITOR_OK
```

procedure to be invoked. The results that will cause the node to be concluded may be specified in either type of node. If no specific results are requested, the node is given a confidence factor of 100 percent as soon as the operation is completed. A sample of the rule-base definition code using these features can be seen in Figure 3.

The ability to reference values returned from procedures without actually causing the procedure to be executed has been provided. In this way, if a procedure has been executed, its return values can be examined. Otherwise, the node is marked false without executing the procedure. This allows the procedural logic for controlling the execution of tests to be separated from the analysis of the results and the resolution of faulty parts, thereby making the rules simpler and easier to understand.

Multiple rule-base segmentation. During the diagnostic process, the amount of memory available for the knowledge base was restricted to approximately 170K bytes of memory. Since each rule node requires 74 bytes and the number of nodes in the rule base exceeds 2400, it is imperative that only those rules relating to components actually present and being tested in the consulting machine be held in memory. At the same time, however, the knowledge base should not need to be changed for any configuration or the addition of any new component. The knowledge base used by the expert system for diagnosing hardware problems on the RT Personal Computer consists of multiple rule-base segments, as shown in Figure 4. The first rule-base segment is the master rule base, which executes procedures to determine the configuration of the machine and then calls device rule-base segments to test the devices that are present. Each device rule-base segment tests one or more of the device options. The entire diagnostic system resides on diskette and includes all rules necessary to test the workstation in any mode.

An action can be associated with any node in a rule base to indicate that the state of the current rule base should be saved, and a new rule base should be paged in. This action will only be taken if the confidence of the current node is evaluated to be greater than the high threshold associated with that node. The new rule base is then traced until all rules have been evaluated.

When all rules in the called rule base are exhausted, the original rule base is reloaded, and the tracing of it is resumed from the point at which it was suspended. Answers gathered in the called rule base and referred to in the original rule base are passed into the original rule base. Goals concluded in the called rule base are appended to the list of goals concluded in the calling rule base. Any number of rule-base calls can be made from any rule base, and a called

rule base in turn may call another rule base as long as there is no recursion. In addition, a rule base is not reentrant.

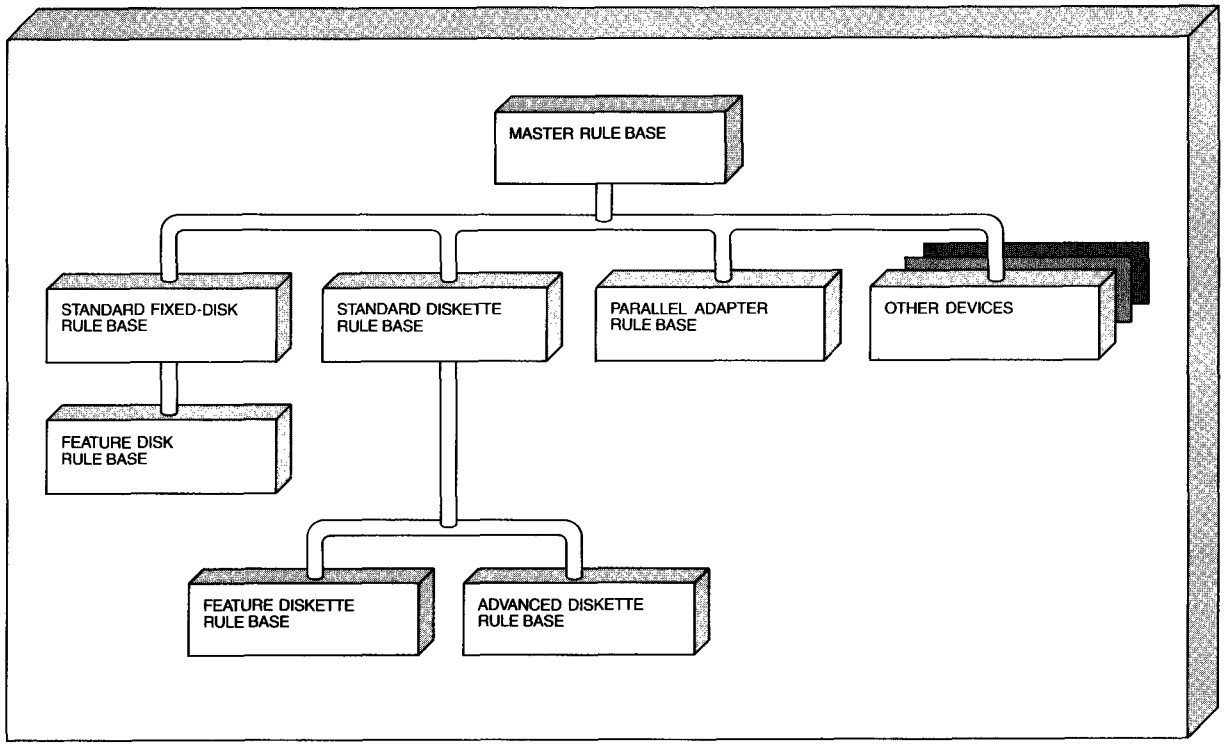
Any evidence that is common to more than one unit can be labeled "GLOBAL." This information is copied onto a global list which is passed between the calling rule base and the rule base it calls. The global list allows information to be passed by more than one rule-base segment.

Besides its usefulness for memory management, the rule-base call has also proved useful for other reasons. Division of the rule base allows it to be separated into segments of coherent knowledge structures. This segmenting makes rule writing and debugging easier and results in a more understandable rule base. Rule-base calls also allow the same rule base to be used several times to conclude goals about similar, yet distinct, items. For example, only one rule base is needed for diagnosing a diskette drive, although a hardware configuration may consist of multiple diskette drives.

Suspending and resuming an inference process. During the course of running hardware diagnostics, it is sometimes necessary to power off the machine. For example, the user may need to power off the machine to replace a part or reseat a card. When the machine is turned back on and diagnostics are resumed, tracing of the rule base should continue at the point where power was turned off. Similarly, if a large or slow program must be executed to obtain data, the rule base can be suspended, the other program executed, and then the execution of the rule base continued after the program has completed.

In order for it to recognize that suspension may occur, any structure that requires the user to power off the machine is given an attribute called "POWER-OFF." If this attribute is encountered by the inference procedure of PIE, control is passed back to the supervising program of PIE before the request is presented to the user. The supervising program saves the information obtained thus far, which includes values obtained from the user, data obtained from external procedures, and concluded goals. It also writes a bit to nonvolatile memory to indicate that resumption will occur. After saving this information, the supervising program returns control to the inference engine, which then presents text to the user informing him or her to turn off the machine.

Figure 4 Organization of rule base segments



When power is turned back on, the resume bit in the nonvolatile memory is checked to see if the expert system is resuming. If so, the supervising program reads in initial information consisting of the name of the rule base that was being processed when suspension occurred and all the saved information about the rule base. The controlling program then calls the inferencing program and passes a parameter indicating that resumption has just occurred. The inferencing program, realizing that the rule base is being restarted from where it left off, does not reset values but uses the information read in by the supervising program. The inferencing program then continues execution from the point at which the interruption occurred.

Implementation

The diagnostic expert system consists of about 16 000 lines of Pascal code. It runs on the RT Personal Computer with at least one megabyte of memory and on a System/370 under VM/CMS.

There are approximately 80 hardware parts that can be identified as faulty by the diagnostic system. The entire knowledge base is composed of more than 30 rule-base segments, each requiring an average of 35 000 bytes of memory. The average rule-base segment contains 80 rule nodes, executes nine test units, and asks five questions. The depth of the trees varies from shallow trees of two levels to very complex trees with as many as thirteen levels. The rule base has a total of 166 GOALs or intermediate hypotheses.

Conclusion

The work reported in this paper has achieved two results. By taking the concepts of segmentation and paging, so common in operating systems, and reapplying these technologies in expert systems, we have lifted size limitations on knowledge bases. Increased size makes it possible to have significant expert-systems capability in desktop computers.

The second result is the demonstration that expert systems can be used to diagnose hardware problems on the machine under test. By dynamically binding procedures during consultation, the consultation module can remain relatively small, yet expand its capability through procedure calls.

Cited references

1. *PC Magazine* **4,** No. 8 (April 16, 1985); entire issue.
2. F. D. Highland, *Design of an Expert System for Shuttle Ground Control,* Master's Thesis, School of Sciences and Technologies, University of Houston, Clear Lake City, Texas (1985).
3. M. T. Harandi, "The architecture of an expert system environment," *Proceedings of the Fifth International Workshop on Expert Systems,* Avignon, France (May 1985), pp. 555–572.
4. P. Nielsen, *A User's Manual for Construct and Consult in the GPSI Environment,* Department of Computer Science, University of Illinois at Urbana-Champaign (1984).
5. M. T. Harandi, "A tree-based knowledge representation scheme for diagnostic expert systems," *Proceedings of the 1984 Conference on Intelligent Systems and Machines,* Rochester, MN (1984), pp. 70–74.

Nancy A. Burns *IBM Engineering Systems Products, 11400 Burnet Road, Austin, Texas 78758.* Mrs. Burns is a senior associate programmer for IBM, currently working with IBM Fellow G. Glenn Henry. She was previously a technical member of the team that developed an expert system to perform diagnostics on the IBM RT Personal Computer. Mrs. Burns received a B.S. in statistics and quantitative methods at Louisiana State University and an M.A. in mathematical sciences at the University of North Florida. She is a member of the Association for Computing Machinery and its Special Interest Group on Artificial Intelligence, the American Association for Artificial Intelligence, and the Association for Computational Linguistics.

T. Jay Ashford *IBM Academic Information Systems, Palo Alto Scientific Center, P.O. Box 10500, Palo Alto, California 94303.* Mr. Ashford was the manager of the Advanced Engineering Systems Diagnostic Control Programs Department. His prior experience includes work in small-systems and I/O architectures, diagnostic and maintenance strategy, and word processing software design. His technical interests include expert systems, operating systems, and systems architecture. He joined IBM in 1974 after receiving the B.A. and M.E.E. degrees from Rice University in 1971. Prior to joining IBM, he worked for the Calspan Corporation in the areas of statistical analysis and estimation. He is currently working at the Palo Alto Scientific Center in expert-systems development.

Christine T. Iwaskiw *IBM Federal Systems Division, 18100 Frederick Pike, Gaithersburg, Maryland 20879.* Ms. Iwaskiw is a senior associate engineer in FSD. She was previously a technical member of the team which developed the hardware diagnostics for the IBM RT Personal Computer. Her experience includes work in power supply development, display hardware diagnostics, graphics software, and expert systems. Her technical interests include expert systems, computer graphics, and systems architecture. She joined IBM in 1981 after receiving a B.E.E. degree from the University of Delaware. Ms. Iwaskiw recently received her master's degree in electrical engineering at the University of Texas in Austin. She is a member of the Institute of Electrical and Electronics Engineers, the Society of Women Engineers, and the American Association for Artificial Intelligence.

Roberta P. Starbird *IBM Engineering Systems Products, 11400 Burnet Road, Austin, Texas 78758.* Ms. Starbird received her B.A. in mathematics from the University of Virginia and her M.A. in mathematics with a minor in computer science from the University of Texas. She joined IBM in 1979, then worked for two and a

half years in Systems Assurance doing performance analysis on small computers. For the next three and a half years, she was the leader of a team working to develop a portable expert system to be used for diagnostics on the IBM RT Personal Computer. She then became the manager of the Hardware System Integration and Test Department.

Richard L. Flagg *IBM Deutschland GmbH, Pueninger Strasse 140, 7000 Stuttgart 80, Germany.* Mr. Flagg received his B.S. in mathematics from Shepherd College in 1964. He joined IBM's Federal Systems Division in 1966. His experience includes pattern analysis, process-control, word processing, communications, and operating systems for diagnostics. He was the technical leader for the advanced Engineering Systems Development Diagnostic Control Programs Department. He is currently a customer service engineer in Stuttgart.

Reprint Order No. G321-5273.