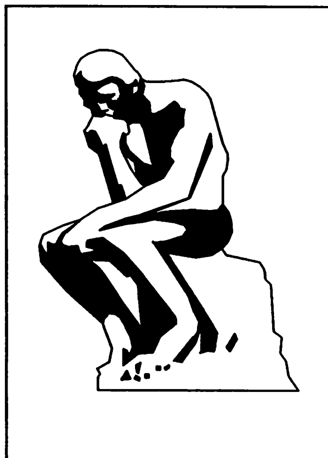
# In Search of a Better Mouse Trap

Dennis Heidner
Boeing Aerospace

## ABSTRACT

This paper covers basic concepts of "expert systems" and their use in business data processing. The author discusses several examples of in-house applications which have been implemented on an HP3000. The cases discussed disprove the beliefs that artificial intelligence systems must be programmed in PROLOG or LISP, and that performance is marginal on a small stack machine.

## INTRODUCTION



Why Artificial Intelligence? The current demographics have been radically changing. The post World War II employees are now nearing the age of retirement. When they retire, years of expertise will leave with them. This loss of knowledge and expertise will have staggering effects on companies that are unprepared. Artificial Intelligence (AI) is an attempt to mimic the human brain with highly processed sand (computer chips) and retain valuable expertise! In their book "Artificial Intelligence: Underlying Assumptions and Basic Objectives" Nick Cercone and Gordon McCalla identify the roots for AI as being *"psychology, philosophy, linguistics, electrical engineering, and computer science"*.[1] (After dabbling in AI for several years I believe that they left off the most important root - sheer madness!) The term AI often causes a considerable debate over what really constitutes a "smart" system. Currently the field which has captured the interest of many researchers is neural nets. Research into neural networks has been as diverse as attempting to simulate neurons in software or developing special hardware which actually uses individual brain cells from slugs and snails. The more traditional areas of AI are expert systems, search and problem solving, theorem proving and logic programming, knowledge representation, learning, and miscellaneous game playing.

One exciting area of AI research is the study of new man-machine interfaces. A considerable amount of attention is now being spent on creating programs which communicate with users on the users' own terms and do not require specialized training. It is possible to buy pocket chess or backgammon games which demonstrate a high level of expertise and yet a very simple and clean man-machine interface. This should be a goal for any expert system. With many newer programs the manual is built-in as part of a very complete help subsystem. The more sophisticated programs even provide context sensitive help; the more errors and more trouble you are in, the more assistance they automatically provide for a specific area.

*Expert systems are computer programs whose behavior duplicates, in some sense, the abilities of a human expert in his/her area of expertise.*[2] Expert programs are considered to be a knowledge base. This knowledge base generally includes any rules of thumb (heuristics) and production rules, facts and relations, and special logic used to present the assertions and questions (the inference engine).

Production rules are similar in appearance to the conditional statements used in existing third generation languages like FORTRAN, BASIC, and COBOL. For example:

```
RULE 1: IF ANIMAL HAS FEATHERS THEN ANIMAL IS BIRD
RULE 2: IF ANIMAL FLIES AND ANIMAL LAYS EGGS THEN ANIMAL IS
        BIRD
```

Facts and relations may be implemented in production rules (as above) or contained within "frames". Frames allow the knowledge engineer to add an hierarchical structure to the knowledge, allow inheritance of traits, and provide slots for data, attributes and rules for interpreting the knowledge. The frame for a bird might look something like:

```
Frame name: Bird
Inherited from: Animal
Slot:  Skin_covering
       Type: Feathers
       Do-procedure: none
Slot:  Reproduction
       Type: Eggs
       Do-procedure: none
Slot:  Extinct_species
       Type: Dodo
       Do-procedure: Take_picture
```

The knowledge acquired by the expert is of little value unless the computer can effectively question and apply the knowledge. This is accomplished with an inference engine. The engine is the "control tower" for the expert system. It is responsible for taking the given facts, applying the assertions and deducing the conclusion. With traditional software systems, the compiler acts as an expert system interpreting the rules (source code), optimizing the object based on expert knowledge, and generating

output suitable for use by the CPU. There are two very basic types of engines. These are the *forward chaining* and *backward chaining* engines.

The forward chaining engine applies the first rule to the known conditions (antecedents); if it is successful then a conclusion (consequent) is accepted. The engine then takes the consequent and looks forward for another rule which may be applied. This chaining of antecedent to consequent and consequent to antecedent continues until there are no more rules to apply or we have reached a goal (the answer). Sometimes it is possible that all rules have been applied (asserted) but a goal has not yet been reached; this is the result of either missing information or inaccurate facts.

A backward chaining engine starts with a goal and, using the known facts, it verifies that the conclusions are supported by the facts. This is accomplished by checking the antecedents in the goal for facts that match. If the facts are unknown then the current antecedent becomes a "sub-goal" (assumption) which the engine tries to prove or disprove. If there are no facts to verify the antecedent then the user is prompted for additional facts.

So far our discussion on the expert system assumes that we live in a perfect world where all facts are known or readily available. However, in the real world, we must often make a decision without all the facts. Experts are able to apply heuristics and make educated guesses for their answers. A good expert system also accommodates missing knowledge or uncertainty.

Missing knowledge or uncertainty in expert systems may be addressed in several ways. The first is to ensure that there is a redundancy of facts. This helps assure that there are fewer holes and provides alternate paths to a solution. A second method is through the use of fuzzy logic. With fuzzy logic there is no longer a simple true or false answer but instead "most likely false", "most likely true," and a large grey area between. When the expert collects the necessary facts to implement the expert system, he/she also tries to determine a default answer and the probability that it will be correct if it must be used in place of an actual fact.

We become experts through a cognitive process in which new facts and techniques are added to our existing knowledge and later reinforced as they are applied to new tasks. Very few expert systems exist which can learn through the same process. Instead, as the knowledge expert becomes aware of deficiencies in the knowledge base, he/she must adapt the knowledge base. The knowledge base for a good expert system is very dynamic. For this reason it is important to involve the end user early in the project, and prototype the system whenever possible. (This is a strong argument for PROLOG and LISP: knowledge prototyping in these languages is immensely easier than in traditional business languages.)

When developing inference engines using COBOL, BASIC, C, SPL, PASCAL or FORTRAN, it is imperative that the engine be simple and modular. Concentrate instead on developing a good user interface, and look for ways to provide redundancy

of facts in your knowledge base. Doing so allows your engine to cope with missing data in an easier manner.

In this paper we will cover three different expert systems. The first is a forward chaining engine, with a discussion about how it was implemented. The second example demonstrates how several independent expert systems can be connected to provide an even smarter system. The third case is a backward chaining inference engine.

## THE SEARCH

Within Boeing Aerospace we have an organization responsible for operating a resource library which contains general purpose test and measurement equipment. The customers (pool users) are the other Boeing organizations which need equipment for engineering design, manufacturing, facilities maintenance, or calibration of other equipment. When a pool user has a need for some test equipment, he/she typically either asks for a specific manufacturer and model number or requests an alternate item which can be used for the test. This search for an alternate model led to the title of this paper - "In Search of a Better Mouse Trap".

Types of test and measurement equipment range from the simple balance scale to highly sophisticated logic analyzers and computer systems. The expert system needs to be able to recommend an alternate for a logic analyzer as quickly as the solution for a scale. The design of the expert system begins by trying to replicate the thought process that an expert instrumentation engineer uses when selecting or looking for an alternate. This process begins with a specific need - "I must weigh a box 4 inches by 5 inches, with a weight between 1 and 5 pounds. The measurement must be accurate to within an ounce." The second step in the process is to identify the manufacturers of scales. The third step is to review the manufacturer literature looking for a scale which is suitable in dimensions, weight range, and accuracy. If step three fails, then a fourth step is to look for alternate weight measuring devices which might be able to perform the measurement with a slightly degraded accuracy.

An early review of our needs identified several basic requirements. First, the expert system for alternates must be very fast (we already had literature libraries and were quite adept at manually locating alternates.) Second, because our computer users varied significantly in education level and typing skills, the user interface must be friendly and easy to use. Third, we want to be able to print a catalog (wish list) containing all the models in our inventory, with specifications, which could be distributed to our pool users. The last requirement is that the engine be flexible enough to support knowledge for many diverse types of test equipment.

### *The Mouse Trap*

The most difficult task in implementing an expert system is acquiring knowledge from experts. Many experts must first visualize a problem before they can begin to solve it. How visualizing helps in resolving problems has been studied in great detail by psychologists studying information processing. Their conclusion is that the knowledge

engineer must "*ask the expert to introspect about the internal processes, to report on inner experience.*"[3] Thus the knowledge engineer must elicit knowledge by asking the expert to describe what thoughts and feelings were used to reach the conclusion. The knowledge engineer must be constantly alert for buzz words which result in missing or uncertainty in the knowledge base. *"References for comparative words like "better," "easier," and "cheaper" sometimes are not qualified. If an expert states: "This is the better system," clarification is required. Appropriate responses are: "How do you know that?" "Better compared to what?" "What, specifically, is better about it?" "Better for what purpose?"*"[4]

The experts that we used to collect our knowledge database were our own internal equipment engineering technical staff. The technical staff comprises four people (including myself) with more than 100 years combined experience in the field of test and measurement equipment. Our knowledge engineers were college students on a summer break or new engineering graduates. The knowledge engineers were given a list of models for which we wanted specifications and the vendors' literature, then encouraged "pick our brains." They were told to restrict the amount of information they amassed to seven or eight of the most critical specifications (COLUMNS) for each class (NOMENCLATURE) of instruments. An early review of the selection process showed that most instruments are selected on the basis of only four or five specifications. We recognized that by restricting the number of fields to only seven or eight there would be instances in which the inference engine could not definitely recommend one model over another. In real life that same uncertainty is often present, even for an experienced instrumentation engineer. Often much more detailed research must be done in order to finally choose an alternate. Therefore, if the expert system is able to reduce the field of items to five or six from one hundred or more, it still can be considered a success!

Each family of instruments is placed into a larger super-class (GEN NOM or CHAPTER). For example BALANCE SCALES, BATHROOM SCALES, and POSTAGE SCALES are all collected into a super-class called SCALES. The knowledge engineers were also requested to identify other classes of instruments which were related or should be checked if a suitable alternate was not found in the original class. (We collected this information in the SEE NOM dataset.)

The collection of super-class (GEN-NOM), class (NOMENCLATURE), and slots (COLUMNS) form a frame, the basis of our expert system. The information for the frame is contained within four separate datasets. The first dataset, called CUSTOMIZE, is an array which contains global information and strategies to be used by the inference engine. An example of a global rule is whether or not the expert engine will allow "vendor loyalty" to be considered when looking for an alternate. The second dataset, called NOMCL-DETL, contains the class name, fields to identify the super-class and any alternate classes, and eight fields which contain pointers to the definitions for up to eight slots. The definitions for the slots, their units and

associated rules are contained in a dataset called NOMH-DETL. The facts which are unique to each vendor's products are contained within a dataset called SPEC-DETL. Our frame of knowledge looks like:

```
              facts                         assertions
------------------------------------------------------------------------------------------
SPEC-DETL: MODEL-CODE, X14
                NOMEN-CODE,  I <=> NOMCL-DETL: NOMEN-CODE,   I
                LINE-NUMBER, I                 NOMENCLATURE, X16 -- class name
                COLUMN1,     R2                GEN-NOM,      I   -- super-class
                COLUMN2,     R2                SEE-NOM,      I   -- alternate class
                COLUMN3,     R2                HEADING1,     I
                COLUMN4,     R2                HEADING2,     I
                COLUMN5,     R2                HEADING3,     I ==> NOMH-DETL: TAG#,      I -- rule#
                COLUMN6,     R2                HEADING4,     I               HEAD_NAME, X8
                COLUMN7,     R2                HEADING5,     I               UNITS,     X8
                COLUMN8,     X8                HEADING6,     I               BETTER-IF, I  ==> global
                                          HEADING7,     I                                scoring rules
                                                                                         from CUSTOMIZE
                                                                                         dataset
```

A requirement for our expert system is that it be easy to use. A brief glance at the knowledge frame above shows that we have several complex relationships which confuse most non-data-processing personnel. It was for this reason and to ease the acquisition and implementation of the expert system that the frame was broken into the four datasets previously shown. Separate data entry routines were provided for each dataset. These routines provided the means to add, delete, or revise facts and assertions. The assertions for the frame must be entered first. The rules are assigned a unique number and may be used by many different frames. A sample dialogue for adding a new rule is:

```
Add/Delete/Revise ? ADD
ENTER TAG NO. 9999
ENTER HEADING WEIGHT
ENTER UNITS POUNDS__
BETTER IF ( =<>X ) ? =
** TRANSACTION COMPLETED **
```

A dialogue revising the assertions for a BALANCE SCALE looks like:

```
Add/Delete/Revise _REV
ENTER NOMENCLATURE BALANCE SCALE______
Working with nomenclature: SCALE,BALANCE
NOMENCLATURE  GEN-NOM  SEE NOM  TOTAL  COL1 COL2 COL3 COL4
SCALE,BALANCE    -120    1515       0     3    2    4    6
ENTER FIELD NAME? HEADINGS_
ENTER COL 1 9999
Heading: WEIGHT     Units: POUNDS
ENTER COL 2 ____
ENTER COL 3 ____
ENTER COL 4 ____
ENTER FIELD NAME ___________
** TRANSACTION COMPLETED **
```

After the rules and class have been established, it is now possible to add the actual facts into the knowledge base. The inference engine bases its decision on the lowest and highest ranges specified by the engineer. A sample dialogue for the acquisition of facts is:

```
Add/Delete/Revise ? ADD
ENTER MFG NONIN
ENTER MODEL INTEREX
ENTER NOMENCLATURE SCALE,BALANCE
ENTER DESCRIPTION It's really neat
ENTER COST 10
Specification line#: 1
ENTER WEIGHT IN POUNDS 1
ENTER Height IN inches
ENTER Width IN inches
ENTER Length IN inches
Specification line#: 2
ENTER WEIGHT IN POUNDS  5
ENTER Height IN inches
ENTER Width IN inches
ENTER Length IN inches
Specification line#: 3
ENTER WEIGHT IN POUNDS
QUIT (Y/N)? Y
** TRANSACTION COMPLETED **
```

Once the knowledge has been entered it may be viewed from any terminal connected to our HP3000. This is an example of what the user sees when viewing the facts just entered.

```
ENTER MFG NONIN
ENTER MODEL interex

MANUFACTURER  MODEL       DESCRIPT         NOMENCLATURE   NEWCOST
NONIN        INTEREX   It's really neat SCALE,BALANCE        $10.

#  WEIGHT   Height   Width    Length
   POUNDS   inches   inches   inches
1   1
2   5
```

*It's Fuzzy*

With the knowledge base assimilated, it is now possible to request the expert system identify alternate make/models. When the inference engine is used, it prompts the inquirer for a manufacturer name and specific model number. The engine retrieves the facts for the specific make/model - then by using the nomenclature code (part of the facts for the model), it connects into the NOMCL-DETL and the NOMH-DETL to load the appropriate assertions. The engine then reads the global rules stored in

the CUSTOMIZE dataset so that it knows how to treat missing facts and what strategy to use. After the assertions have been loaded onto the stack, the engine begins to examine other make/models in the same class (nomenclature) as the one specified by the inquirer. Our engine uses the forward chaining technique for applying the assertions that it has loaded. The potential capability of each model is "scored" using a point system. Only the scoring results for the top one hundred make/models are saved. While examining each model the reason why each has received its high marks is saved with the score. Later when we display the results we can also display the reason why.

In the score card that we use, a perfect match is worth 100 points and is indicated as "GOOD." A near match (within 10% of the desired range) is worth 80 points and remembered as a "FAIR" match. An instrument which comes within 20% of our original instrument is considered a "POOR" match and receives only 60 points. An instrument which falls short is given no points for the specific slot unless the slot is empty because of missing information. In this case 10 points is assigned. Each slot has a multiplier associated with it. The first column, considered to be the most important, is worth five times more points then the last column. This results in the following point system:

| Column | Multiplier |
|---|---|
| 1 | 6 |
| 2 | 5 |
| 3 | 4 |
| 4 | 3 |
| 5 | 2 |
| 6 | 1.5 |
| 7 | 1 |

The thresholds for poor, fair, good as well as the multiplier values are stored in the global rule set (CUSTOMIZE) and can be adjusted independently of the facts in the SPEC-DETL or the assertions in the NOMCL-DETL or NOMH-DETL.

*Eeek! A mouse!*

The visual format for product comparisons is generally a table which lists the specifications and features for each selection. Comparisons of this type are seen in almost every magazine and many newspaper advertisements. Because this is an accepted method for comparisons with which most people are already familiar, it was the method that we chose for the inference engine's presentation of equipment alternates. The only information that the engine needs from the inquirer is a manufacturer name and the model number. For example:

```
ENTER MFG Heidner
ENTER MODEL B723c
There are 2  Heidner B723C
MANUFACTURE   MODEL      DESCRIPT          NOMENCLATURE  NEWCOST  NEWDATE
 Heidner     B723C     100Hz-22GHz       ANLYZ,SPECT     $55,555. 01/01/89
NATIONAL STOCK NUMBER:       - - -

#  Freq      Resoltn  Shape Ft MaxAC In  In Pwr    In Accu   Swp Time Unit
   Hz        Hz       Type      dbm      dBm       +/- dB    Sec      Requires
1   100       10       15        30       -134      .6        1.0u    Stand
2   22G       3000K    11                 30        3         1500    Alone
ME, LP, NITE, QUIT? ME
```

The engine first displays the specifications of the item for which you are seeking an alternate. The inquirer is allowed to choose whether the results of the search are listed to the CRT (ME), sent to the line printer (LP),or submitted as a job to be run at night (NITE); to quit altogether, he/she enters QUIT. Choosing ME causes the following to be displayed.

```
ALT FOR         COL1      COL2      COL3     COL4      COL5      COL6      COL7
 Heidner        Freq      Resoltn   Shape Ft MaxAC In  In Pwr    In Accu   Swp Time
 B723C          Hz        Hz        Type      dbm      dBm       +/- dB    Sec

Z#   MFG        MODEL       COL1  COL2  COL3  COL4  COL5  COL6  COL7
-------------------------------------------------------------------
 1   ELKTRONIX  8M12              GOOD
 2   ELKTRONIX  8M13                    GOOD  GOOD  GOOD
 3   ELKTRONIX  8M18                    GOOD  GOOD  GOOD
 4   ELKTRONIX  892A                    GOOD  GOOD
 5   ELKTRONIX  8M5                     GOOD  GOOD
 6   ELKTRONIX  892P                    GOOD  GOOD
 7   ELKTRONIX  892                     GOOD  GOOD
 8   ELKTRONIX  894P                    GOOD  GOOD
 9   ELKTRONIX  8M14                    GOOD
Press [RETURN] to go on, ZOOM number or 'QUIT' to stop: Zoom 1
```

The engine then proceeds to display the more detailed specifications for the item chosen by the inquirer.

```
MANUFACTURER  MODEL       DESCRIPT          NOMENCLATURE  NEWCOST  NEWDATE
 ELKTRONIX  8M12      2GHz ELK8000 PI   ANLYZ,SPECT      $5,000. 01/01/73
NATIONAL STOCK NUMBER:       - - -

#  Freq      Resoltn  Shape Ft MaxAC In  In Pwr    In Accu   Swp Time Unit
   Hz        Hz       Type      dbm      dBm       +/- dB    Sec      Requires
1   100K      300                                                     8000
2   1800M     3000K                                                   Mframe
USE WITH:8813     8803     8854
```

The software required to collect or edit the facts and assertions requires three code segments, each about 6K words. The inference engine requires two segments each approximately 8K words long. The stack size used is approximately 5K words. This is quite small compared to what would be expected for a PROLOG or LISP implementation. However, in exchange for the smaller size and faster execution, we were required to forego the extra flexibility offered by PROLOG or LISP.

## THE SCROUNGER

The test equipment pool has almost one thousand customers who use tens of thousands of items spread throughout western Washington state. The equipment pool is very dynamic with equipment moving approximately every 50 days. As a resource organization we are committed to serve our customers with fastest possible response time at the lowest possible cost. This requires the assistance of master "scroungers" (like Radar O'Riley from the television program "M.A.S.H.") Radar's uncanny ability to "perceive" events about to happen provided him with an edge he needed to cope with the war, perform his routine activities, and maintain the stock of supplies for the unit. The ideal expert system "scrounger" should have many of these same attributes. The scrounger must be able to find "who's got it", ask "can I get it?", and then try to "grab it!"

### *Who's got it?*

The location of all the test equipment in the equipment pool is contained within a TurboIMAGE database on our HP3000. With access to this database the scrounger is able to locate all items. That is the easy part! If we have 500 or 600 items of one make/model, we do not want to make 500 or 600 telephone calls while trying to negotiate a loan or swap of an item. (This is even more important if there is one sitting in a stockroom someplace!) We have solved this problem by implementing an expert system which locates the items and assigns a probability that we can borrow or loan the item out. The procedure RATEQUIP is a forward chaining inference engine that also loads rules and strategy from the CUSTOMIZE dataset. The engine examines all items of the specific make/model, checking to see if the item is currently assigned to a user, if it has been reserved for a future test, if the current schedule is about to expire, if the current user of the item has many like items, and whether or not the current user has a history of not fully using the equipment. RATEQUIP returns the asset identification numbers and the score to the procedure FINDMFGMODEL. If an inquirer wishes to see what is available, FINDMFGMODEL displays more detailed information, with the item which should be picked first at the top of the list.

### *Can I get it?*

After a list of potential items has been created, it can then be passed on to the next expert system. This next system uses an engine called CHECKSCHEDULE. CHECKSCHEDULE takes each item and looks to see if we can "squeeze" another use in for this item. If we cannot, CHECKSCHEDULE returns an error, along with the date when the item will be available. If the item is available, then a flag is returned to indicate we can proceed to schedule this item.

### *Grab it!*

The function which grabs the items has not yet been completed. The procedures have been tried manually and are straightforward: take an item which can be scheduled, and lock it in.

*Trade?*

Remember the real world? More often than not, the specific item is not available to be "grabbed". In that case, the inference engine from our specification expert system is invoked to identify other possible models which could be used. The engine SCOREMODEL returns a list of up to 100 possible alternates. Each model can, in turn, be sent back to the RATEQUIP engine which provides a list of items with the most likely to be loaned at the top of the list. CHECKSCHEDULE is invoked for each item, those which fail are dropped off the list. Finally the several remaining items are passed on to the "Grab it" process with the request that soft schedules be placed on them. ("Soft" means that a firm request for that specific model will take precedence.) The "Grab it" process is instructed to send an electronic mail message to the inquirer who requested the make/model explaining what has been done.

Will it work? The modules RATEQUIP, SCOREMODEL, CHECKSCHEDULE, FINDMFGMODEL, and SENDMAIL have been in production for several years. They are fast and trouble free. Of all the expert systems discussed so far, the "grab it" module appears to be almost trivial. The concept of coupling the engines together is not new - we "experts" currently perform related tasks by hand every day.

## THE DOCTOR

Controlling the cost of software development and maintenance is a major concern of data processing managers. Software maintenance is labor-intensive work. Anything which can be done to improve the productivity of the maintainer will, in turn, reduce support costs. One technique that we have employed is extensive built-in diagnostics and debugging tools. The programs have been written so when an error is detected, information is collected which will assist the maintainer in quickly locating and correcting the problem.[5] A special program called ADPAN reads the information "snapshot" and attempts to identify where the error occurred.[6] ADPAN acts as a specialized "doctor" for software. In the process of diagnosing the error ADPAN must locate the last valid stack marker, play the part of the MPE loader and locate all subroutines which created stack markers, look for transitions between user-code and system-code, check trap Plabels, and finally check the status of all files open at the time of the snapshot. ADPAN contains several inference engines which cooperate while making the diagnosis. We will concentrate on the simplest, the process used to find the stack markers.

When we first began implementing snapshots into our programs, we used the MPE intrinsic STACKDUMP to collect and format the process stack. This originally provided us with a solution which did not require privilege mode and was supported by H-P. When a program encountered an error, it entered a procedure which opened a snapshot file, then directed STACKDUMP to copy the entire stack into it and format the program's stack markers. Later an analyst could simply use FCOPY to print the snapshot. Reading the formatted markers required some effort, but at least

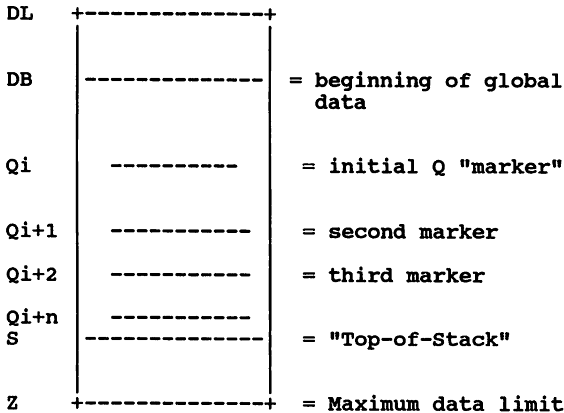
there was documentation on how to interpret them[7]. That was life with MPE III and early versions of MPE IV.

Then sometime around MPE IV D-MIT, several bugs were introduced with new versions of the operating system. The first was somewhat humorous (but a significant security risk). STACKDUMP allowed the calling procedure to specify a range to dump. If the range specified was beyond what the user really had - STACKDUMP should dump only up to the limits of the user's stack. The new undocumented "feature" instead dumped the entire 64K word memory page, complete with other users' data and passwords! After reporting the problem H-P provided a patch. The patch corrected the security problem; however, the stack markers were no longer formatted!

Without formatted markers the post mortem dumps appeared to be worthless, until we realized that we could manually look for the pattern which appeared to be a marker and verify it by hand-tracing back to the initial marker. This required some effort and was difficult to teach. Unfortunately the new version of MPE which would contain the correction to STACKDUMP (Q-MIT) was months away. After a little deliberation we decided to write a program which would replicate what we were doing manually. This program later became the basis for our backward chaining inference engine.

The stack for the "classic" HP3000 contains a dynamic region between "DL" and "DB"; the area above DB is called the global area and ranges from DB up to the first stack marker Qi. Every call to a subroutine (or COBOL PERFORM) causes a new stack marker to be created and added to the stack. The stack on the HP3000 appears to "grow" downward until a maximum limit of Z has been reached. The "Top-of-Stack" (TOS, shown at the bottom of the figure) is called S. The procedure's dynamic local variables are located between the last stack marker and S.

Stack markers have four special values in them. The first, located at Q, is called displacement (or delta). The next stack marker can be located by taking the current address location of Q and subtracting the displacement from it. The second word (Q-1) contains the status of the CPU at the time of the subroutine call. The third word (Q-2) is called the "P_Relative" value. P_Relative is the word location in the program to which the program will return when the current subroutine completes. The last word (Q-3) is an index register. This word can contain any value. For the experienced gurus out there, this may be quite boring; however from the neophytes, I can already hear the demands to stop. It should be now apparent why we want an expert system to assist us in locating the markers.




The method we use to find all the stack markers is:

1. Start by checking the value at S. If it is not the marker then decrement the address count by 1 and check this new value.

2. We can prove or disprove that word is Q in a stack marker by applying the following rules.

   A. The Delta Q must always be a positive number greater than 4 but less than the address location of Q.

   B. Check P_Relative. It must range between 0 and 16384.

   C. The value of STATUS (Q-1) may not be 0.

   D. The value pointed to by the address of Q minus the displacement must be a stack marker.

   E. If Q=4 and P_Relative=0, we are at Qi and have located all the stack markers.

We can more clearly see why this inference engine which locates the markers is considered to be backward chaining by carefully looking at rule 2D. This requires we not pass judgement on the starting value we are examining until we have chained backward and verified each "sub-goal". The FORTRAN source code which is used to implement the inference engine is listed next.

```
      SUBROUTINE FIND MARKERS(Q,IDUMPFILE,START,END,LASTREC,DB OFFSET,S,INBUF,IERR,MPE V)
C
C     The purpose of this routine is to try to find the stack markers in the dump file, when the marker
C       display has been damaged.
C
C     Error codes returned are:
C     IERR=0  - OK
C     IERR=-1 - Problems encountered trying to read dumpfile
C     IERR=1  - Unable to locate a good marker!
C
      INTEGER*4 LASTREC,START,END
      INTEGER Q,S,IDUMPFILE,IERR,Q TEST,DBOFFSET
      LOGICAL INBUF(128),MPE V
C
C     The algorithm used to find a valid marker is as follows.
C
C     1.  Set the absolute maximum number are words we will try to 2048.
C     2.  Set  Q TEST = S
      Q TEST = S
C
C     3.  VERIFY MARKER using the value of Q TEST. If ok then IERR = 0, and Q = Q TEST, then RETURN.
C         Else....
C
      DO 100 I=1,2048
      CALL VERIFY MARKER(Q TEST,IDUMPFILE,START,END,LASTREC,DB OFFSET,S,INBUF,IERR,MPE V)
      IF(IERR.EQ.0) GOTO 200
C
C     4.  Decrement Q TEST.
C
      Q TEST = Q TEST - 1
C
C     5.  Proceed to step 3.
C
100   CONTINUE
      IERR=-9
      RETURN
C
C     WE FOUND A GOOD MARKER
C
200   Q = Q TEST
      RETURN
      END
```

```
      SUBROUTINE VERIFY MARKER(Q,IDUMPFILE,START,END,LASTREC,DB OFFSET,S,INBUF,IERR,MPE V)
C
C     The purpose of this routine is to take a given value
C      for Q and prove or disprove that it is a stack marker.
C
C     Error codes returned are:
C     IERR = 0    Okay. Hypothesis proven
C     IERR = 1    Bad marker, Hypothesis is false
C     IERR =-1    Problems encountered in GETWORD, Hypothesis false
C
      INTEGER*4 START,END,LASTREC
      INTEGER DB OFFSET,Q,IDUMPFILE,IERR,S,Q TEST,P RELATIVE,DELTA Q
      INTEGER STATUS
      LOGICAL INBUF(128),MPE V
 C
       Q TEST = Q
 C     The rules for checking for a valid stack marker are:
 C
 C     1.  Take the value of Q it must fall between DB OFFSET and S.
 C         If not then IERR=1 and return.  If ok proceed...
 100   IERR=1
       IF((Q TEST.LT.DB OFFSET).OR.(Q TEST.GT.S)) RETURN
 C
 C     2.  GETWORD at address of Q. If error encountered in GETWORD then IERR=-1, and return
 C         If ok  proceed....
 C
       IERR=-1
       CALL GETWORD(Q TEST,DELTA Q,IDUMPFILE,START,END,LASTREC,DB OFFSET,INBUF,NERR)
       IF(NERR.NE.0) RETURN
 C
 C     3.  Check value at address of Q, it must be in range of 4 to address of Q.
 C         If not then IERR=1 and return. If okay proceed...
 C
       IERR=1
       IF((DELTA Q .LT.4).OR.(DELTA Q.GT.Q TEST)) RETURN
 C
 C     4.  GETWORD at Q-2.  This is P'RELATIVE.   The value for P'RELATIVE must be
 C    between 0 and 16384.  If not then IERR=1, and return.  Else proceed....
 C
       IERR=-1
       CALL GETWORD(Q TEST-2,P RELATIVE,IDUMPFILE,START,END,LASTREC,DB OFFSET,INBUF,NERR)
       IF(NERR.NE.0) RETURN
       IERR=1
       MPE V = .FALSE.
       IF ( P RELATIVE .EQ. %40000 ) MPE V = .TRUE.
       P RELATIVE [0:2] = 0
 C
 C     4B. GETWORD at Q-1.  This is STATUS. The value for STATUS must not be zero!  If it is then IERR=1,
 C         and return.  Else proceed....
 C
       IERR=-1
       CALL GETWORD(Q TEST-1,STATUS,IDUMPFILE,START,END,LASTREC,DB OFFSET,INBUF,NERR)
       IF(NERR.NE.0) RETURN
       IF ( STATUS[8:8] .EQ. 0) RETURN
 C
 C     5.  Save value at Q   Derive  deltaQ. Take Q and subtract value  at Q from Q.
 C
       Q TEST = Q TEST - DELTA Q
 C
 C     6.  If value of Q = 4  and  P'RELATIVE = 0  then  IERR=0 and RETURN!   (WE HAVE FOUND Qintial).
 C
       IERR=0
       IF((DELTA Q .EQ. 4).AND.(P RELATIVE.EQ.0)) RETURN
 C
 C     7.  Use the value of the Q - delta Q and a new address for Q.  Then proceed to step 1.
 C
       GOTO 100
       END
```

## CONCLUSION

Dr. David Hu characterizes the expert system as:

> "* *An expert system mimics experts or specialists in a specific field - for example, medicine or computer configuration.*
>
> * *The power of an expert system lies in knowledge and how it is represented, not in programming technique.*
>
> * *The principal components of current systems are knowledge base, inference engine, and man-machine interface.*
>
> * *The knowledge base contains facts and rules that embody an expert's expertise.*
>
> * *The three commonly used methods for encoding facts and relationships that constitute knowledge are rules, frames, and logical expressions.*
>
> * *Inference engines are relatively simple. The most commonly used methods are backward chaining and forward chaining.*
>
> * *User interface is a weak but critical element of expert systems. Many current expert systems are equipped with "menus" and explanation modules to allow users to query expert systems and examine their output statements"[8]*

We have seen three examples of applications which are expert systems in their narrow field. These systems are been written in the traditional third generation languages of FORTRAN and SPL. They are fast, use little stack and code space, yet provide functions which otherwise would consume an expert's time. In the first two applications the database management system is TurboIMAGE and the third is accomplished using ordinary MPE file I/O.

## REFERENCES


[1] Cercone, Nick and McCalla, Gordon, *Artificial Intelligence: Underlying Assumptions and Basic Objectives, from Journal of the American Society for Information Science,* Volume 35, Number 5 September 1984 pp 280-290

[2] ibid.

[3] Evanson, Steven E., *How to TALK to an EXPERT, AI EXPERT,* February 1988 pages 36-41

[4] ibid.

[5] Heidner, Dennis L., *The Bug Stops Here,* Paper presented at the 1987 HP3000 International Users Group Conference, Las Vegas, Nevada

[6] ADPAN is in the public domain contributed software library available from INTEREX, 680 Almanor Ave. P.O. Box 3439, Sunnyvale, California, 94088-3489

[7] Hewlett-Packard, *MPE Debug/Stack Dump reference Manual*, part number 30000-90012

[8] Hu, David, *Programmer's Reference Guide to Expert Systems*, (Indianapolis, IN: Howard W. Sams & Co., 1987) page 10

## BIOGRAPHY

Dennis Heidner received his BSEE degree from Montana State University, Bozeman, Montana. Mr. Heidner has written and presented numerous papers at the HP International User Group Conferences. Mr. Heidner is a co-author of *The IMAGE/3000 Handbook* and the *TurboIMAGE Supplement* published by WordWare, Seattle, Washington. He has written technical articles which have been published in several magazines. Mr. Heidner is a member of the Association for Computing Machinery (ACM) and the Institute for Electrical and Electronic Engineers (IEEE).