EXPERT SYSTEM FOR COST VARIANCE INVESTIGATION

By

Jayanth Rama Varma

W P No. 895
September 1990

INDIAN INSTITUTE OF MANAGEMENT
AHMEDABAD-380 015
INDIA

EXPERT SYSTEM FOR COST VARIANCE INVESTIGATION

ABSTRACT

Jayanth Rama Varma

In a typical standard costing or budgetary control system, a manager might receive a variance analysis reporting several hundred variances, of which many may have arisen due to random factors, or may be too insignificant to merit attention. The manager uses his knowledge and experience to identify the important variances which demand further investigation. Both management accounting theory and statistical decision theory can make significant contributions towards improving this decision, but both make extravagant demands on the manager in terms of the theoretical and factual knowledge expected of him. Much of this knowledge, even if available, is scattered throughout the organization with very little readily accessible to top management itself. This research takes the view that a Knowledge Base / Expert System approach can be useful in this context.

An expert system was implemented in Turbo Prolog using fuzzy logic and MYCIN type certainty factors to handle uncertainty. Though traditional Prolog interpreters can be used directly to write Expert Systems without using an Expert System Shell, this is not the case with the Turbo Prolog compiler. It becomes necessary to write an interpreter/Expert System Shell in Turbo Prolog using some of the software tools (scanner and parser)

available along with the Turbo Prolog compiler itself.

The expert system was tried out on a case on cost variance investigation from a well known book on management control systems (Antony, Dearden and Bedford, 1984). The substantive performance of the system in this armchair case study was quite encouraging.

In terms of speed and memory requirements, the system is close to the limits of what is possible in the PC environment with Turbo Prolog. It is likely that further work in this area will have to move out of the PCs to the workstations or to other more powerful computing platforms.

The most important enhancement that is needed in the current system is a natural language interface; the current Prolog-like interface is acceptable only in classroom/research settings.

The system has had considerable success in its principal research objectives. However, on the question of integrating statistical decision theory with fuzzy logic and certainty factors, the expert system methodology appears to be at a dead end; perhaps real progress in this area will come from purely statistical approaches to the problem.

A C K N O W L E D G E M E N T S

The work reported in this paper was carried out under a research project of the Computer Aided Management (CAM) Centre at the Indian Institute of Management, Ahmedabad. The resource support provided by the CAM Centre is gratefully acknowledged.

CONTENTS

1. THEORETICAL FOUNDATIONS

1.0 Introduction

Standard Costing and Budgetary Control are among the most important techniques of cost control in modern business. In these systems, actual costs are compared with the standard or budgeted costs, and the variance in the costs provides a signal for corrective action.

Of course, not all variances need such corrective action. Many variances arise from random factors, or are too insignificant to merit attention. The manager who receives a variance analysis reporting possibly several hundred variances has to pick out the important variances to be investigated. Managers usually make this judgment on the basis of their knowledge and experience (Horngren, 1975).

1.1 Statistical Decision Theory

In principle, this problem can be solved by using statistical decision theory (see, for example, Dittman & Prakash,(1978)). The theory of sequential decisions provides the concepts of the Expected Value of Perfect Information (EVPI) and the Expected Value of Imperfect (Sample) Information (EVSI). These concepts can be applied here if we can quantify:

1. the probability that the variance is due to a genuine problem rather than to random factors.
2. the costs of the investigation.
3. the benefits that will arise from the investigation and the consequent corrective action if any.

In practice, this quantification is quite impractical.

Quantification of the probabilities requires modeling several complex and interrelated stochastic processes which are only imperfectly known. The costs of the investigation consist mainly of top managerial time which is difficult to estimate beforehand. As regards the possible benefits from the investigation, the impact on short run profits can often be estimated. Managers however are interested in several Key Result Areas and not just short term profits. It is true that these multiple objectives can be subsumed under the rubric of long run profit maximization, but this concept again is not easily operationalizable.

Moreover, the manager must also consider how his implementation of the control system will affect the behaviour and motivation of those whose performance he is monitoring. In some closely related problems, this game theoretic aspect has also been successfully modeled (Averhans & Frick (1977) and Frick (1977)), but in most realistic situations, the game theoretic formulation is intractable.

Because of these difficulties in mathematical modeling, the models have found very little application in actual decision making (Kaplan, 1975; and Demski & Kreps, 1980)). We must recognize however that while these methods are unable to solve real life problems, they provide valuable insights which can supplement the "knowledge and experience" on which managers rely.

1.2 Accounting Theory

Management accounting theorists have developed an elaborate theory of variance analysis and variance decomposition. The principal objective of this theory has been to pinpoint a variance to the lowest responsibility centre which has caused the variance. In the hands of skilled practitioners, this is a very potent weapon, but to the unskilled user the most visible impact of the management accountant is a further proliferation of variances in the report that he receives. Traditionally, accountants have tried to mitigate this problem by better design of accounting reports, preparation of summary reports and verbal explanations. None of this is, however, an adequate substitute for a sound knowledge of the accounting theory from which they derive their rationale. Two obvious possibilities that come to mind are :

1. To provide training in accounting theory and principles to the managers involved.
2. To let the accountants carry out the follow up of variances themselves, substituting their judgment and analysis for that of the manager.

Since neither is a very satisfactory solution, the question that arises is whether it is possible to do something better.

1.3 Fragmentation of Knowledge

The above discussion implies that the manager looking at a variance report needs to utilize a substantial amount of background knowledge to make meaningful decisions. This knowledge includes knowledge about any of the following:

1. Costing and management control theory.
2. Statistical decision theory.
3. Technology and environment of the organization.
4. Skills and attitudes of the people in the organization.

All this knowledge is usually scattered throughout the organization, and only a small part is available with top management itself. The first task is to provide the remaining knowledge to the manager in a readily assimilable form. The second task is to facilitate the integration of this knowledge with the manager's private knowledge to arrive at a valid conclusion.

1.4 The Knowledge Base or Expert System Approach

The approach adopted in this research is that the problem of fragmented knowledge outlined above can be solved by developing formalized, shared knowledge bases within the organization. A computer based Expert System would then be able to access these different knowledge bases and integrate them using its in-built inference ability.

Traditional application of computers in the domain of accounting have been more concerned with creating comprehensive data bases, and providing the user with the ability to integrate data from diverse sources. What is being suggested here is that it is similarly necessary to create comprehensive knowledge bases which formalize the knowledge scattered throughout the organization, and make it shareable. The computer system should then be able to integrate this public, shared knowledge with the manager's own private, nonshared knowledge through its inference ability.

This leads to two main research issues:

1. How can the "knowledge and experience" of the managers be formalized, generalized and shared within the organization?
2. How can the theoretical insights and heuristics suggested by accounting theory, statistical analysis and mathematical modeling be integrated with the knowledge of the managers as so formalized?

This research addresses these questions within the context of current Personal Computer capabilities. It is an exploratory attempt to assess the applicability of Expert System techniques and methods to the domain of cost variance investigation.

2 Reasoning under Uncertainty

2.0 Introduction

One thing which emerges clearly from the theoretical analysis of the cost variance investigation problem is that it is essentially statistical in nature. Any attempt to tackle it would therefore involve inference under uncertainty.

There are three main approaches to handling uncertainty in inference systems :

1. Bayesian inference
2. Fuzzy reasoning
3. Hybrid systems like MYCIN

2.1 Bayesian Approach

Of the three approaches mentioned above, the Bayesian approach is the one with the strongest theoretical underpinnings, viz., Bayes theorem in probability theory. We first restate the Bayesian revision formulas in terms of odds rather than probabilities. (A probability of **p** is stated as an odds of **p/(1-p)**; e.g., a probability of 0.10 becomes an odds of 1:9 or 1/9). Let **e** denote any piece of evidence and **h** the hypothesis being investigated. Let ~**h** (the negation of **h**) be the alternate hypothesis. By Bayes theorem, we have

P(h|e)P(e) = P(eh) = P(e|h)P(h).

Hence, P(h|e) = P(e|h)P(h)/P(a).

Similarly, we have

$P(\sim h|e) = P(e|\sim h)P(\sim h)/P(e)$

Dividing, we obtain

$P(h|e)/P(\sim h|e) = [P(e|h)/P(e|\sim h)] [P(h)/P(\sim h)]$

The LHS is the posterior odds (i.e. the odds in favour of **h** after taking into account the new evidence), and the second factor on the RHS is the prior odds (i.e., the odds in favour of **h** before the new evidence is considered). **P(e|h)** is the likelihood of the evidence **e** under **h**; **P(e|~h)** is the likelihood under **~h**. The first factor on the RHS is thus the likelihood ratio. This establishes the formula that the posterior odds equals the prior odds times the likelihood ratio.

Inference using Bayesian approach is, therefore, conceptually quite simple : given a new piece of evidence **e** we revise our odds for the hypothesis **h** by multiplying the odds by a positive factor -- the likelihood ratio. If the factor exceeds unity, evidence favors **h** and the odds in favour of **h** improve; if it is less than unity, the evidence is against **h** and the odds decline. A likelihood ratio of zero disproves the hypothesis completely, and a ratio of infinity proves it completely. A Bayesian inference system will, therefore, consist of a number of rules each of which links an evidence **e** to a hypothesis **h** with a likelihood ratio which specifies the direction and strength of the relationship.

There are two main practical difficulties with the approach :

1. Multiple rules involving the same hypothesis can be applied separately only if the evidences they rely on are independent. For example, if we have a rule linking evidence e1 to h with an LR of 2.5 and another rule linking e2 to h with a ratio of 3, we would be tempted to use both rules one after the other to multiply the odds for h by 2.5 * 3 = 7.5. This is valid if and only if e1 and e2 constitute independent pieces of evidence. Otherwise, there is the danger that we would be double counting some of the evidence. Checking for independence statistically is not difficult if sufficiently large data is available; but in the cost variance application that we have in mind, the requisite data is not available, and reliance has to be placed on the judgment of the user.
2. In the absence of enough data to permit a statistical estimation of the likelihood ratio, this has to be estimated by the user on the basis of his judgment. Except to those who have a thorough grounding in Bayesian statistics, the likelihood ratio is a number devoid of intuitive meaning; to most people it becomes a mere number measuring the strength of association. The likelihood ratio is not very suitable for this kind of use because of the inconvenient range of this number — zero to infinity. Most people find it easier to state their measures of uncertainty on a zero to one scale. The other inconvenient feature of the likelihood ratio is its asymmetric scaling of favorable evidence (one to infinity) and unfavorable evidence (zero to one).

Keeping in view the fact that the expert system would have to be used mainly by people with less than a perfect understanding of Bayesian statistics, it was decided not to adopt the Bayesian approach in this work. This was done despite my personal belief that the Bayesian approach is the only theoretically valid model for reasoning under uncertainty, and that any system that departs from the Bayesian framework is to that extent wrong. The only possible justification for any nonbayesian method is that it provides a workable and reasonable approximation to the correct Bayesian method.

2.2 Fuzzy Reasoning

While classical logic asserts that any proposition is either true or false, fuzzy logic is based on the idea that a proposition can have a variety of truth values ranging from zero to unity. A truth value of zero represents classical falsity, unity represents classical truth, and intermediate values represent fuzziness. Fuzzy logic like classical logic has rules for computing the truth value of complex propositions from the truth values of their constituent propositions. In every case they reduce to the classical rules for truth values of zero and unity :

```
truth_value(p and q) = min(truth_value(p), truth_value(q))
truth_value(p or q)  = max(truth_value(p), truth_value(q))
truth_value(not p)   = 1 - truth_value(p)
```

It should be emphasized that the fuzziness cannot be interpreted as probability because probabilities do not obey the above formulas. For example, probability of (**p and q**) is not equal to the minimum of the probabilities of **p** and **q**; if **p** and **q** are independent, the two probabilities have to be multiplied giving a probability which is strictly less than the probability of either alone.

The main attraction of fuzzy logic is its simple and fast method of combining various pieces of evidence to make inferences. In classical logic, when **p** is known to be true, we can use a rule "if **p** then **q**" to infer **q** is true. In fuzzy logic, when we know the truth value of **p** we can use the rule "if **p** then **q**" to attach the same truth value to **q** also. If the rule is "if **p** and **q** then **r**", we must first compute the minimum of the truth value of **p** and **q**, and attach this to **r**. If we have two different rules "if **p** then **r**" and "if **q** then **r**", we can consider this as equivalent to "if **p** or **q** then **r**" and use the maximum of the truth values of **p** and **q**.

In the above examples, we assumed that the rule "if **p** then **q**" was itself known with certainty, but, in general, this itself could be a fuzzy rule, i.e., there could be fuzziness attached to the rule itself. In this case, we could take the minimum of the truth value of the premises of the rule (i.e. **p**) and the truth value of the rule itself, and treat this as the truth value of the conclusion (i.e. q).

Despite its simplicity, the drastic departure of fuzzy reasoning from the tenets of probability theory make its use slightly problematic. The more serious problem is that there is no way in which the system can take several independent corroborating pieces of evidence to provide a high truth value for the conclusion. In the fuzzy system, the truth value of the conclusion is simply the maximum of that obtained from applying each rule separately. A collection of rules is only as good as its strongest rule. This is a major shortcoming because, in reality, we often come across several pieces of confirming evidence each of which is individually weak but collectively establish a very strong case. A good expert system must be able to deal with this situation.

Though the pure fuzzy system is, therefore, unacceptable, it is, nevertheless, a major component of the hybrid systems like MYCIN which have been practically successful. Moreover, the fuzzy system is the most practical method of operationalizing many of the fuzzy concepts of everyday language. For example, what does it mean to say that a variance is large? Classical logic would require a definition of the form a variance is large if it exceeds x%. The classical law of the excluded middle would assert that a variance is either large or not, and there should be no ambiguity at all on this score. Fuzzy logic would however allow us to retain the vagueness that is inherent in the intuitive notion of largeness. Fuzzy logic would not insist on an yes-or-no answer to the question whether a variance is large; it would permit various shades of certainty in the answer which are captured in the form of a number between 0 and 1. This vagueness is extremely useful in formulating rules for the cost variance investigation problem.

2.3 Hybrid (MYCIN type) Systems

The main contribution of the MYCIN system (Shortliffe and Buchanan, 1984) is a method of combining several pieces of evidence relating to a hypothesis. Suppose there are several pieces of evidence, e_1, e_2, ..., e_n, which when used separately yield "Measures of Belief", $MB[h;e_1]$, $MB[h;e_2]$, ..., $MB[h;e_n]$, respectively. In other words, $MB[h;e_k]$, represents the degree of belief in **h** induced by the k'th piece of evidence e_k. MYCIN combines these values into a single measure of belief $MB[h;e_1,e_2,...,e_n]$ which satisfies :

$$(1 - MB[h;e_1,e_2,...,e_n]) = (1 - MB[h;e_1]) * (1 - MB[h;e_2]) * ... * (1 - MB[h;e_n])$$

OR

$$MB[h;e_1,e_2,...,e_n] = 1 - (1 - MB[h;e_1]) * (1 - MB[h;e_2]) * ... * (1 - MB[h;e_n])$$

One can also perform this computation incrementally by setting MB to zero initially and then applying each rule by turn updating MB by the formula :

$(1 - MB_{new}) = (1 - MB_{old}) * (1 - MB[h;e_k])$

OR

$MB_{new} = 1 - (1 - MB_{old}) * (1 - MB[h;e_k])$

The MYCIN system uses only evidence supporting the conclusion to compute MB. It combines all evidence against the conclusion to compute a Measure of Disbelief (MD) using formulas similar to those for MB. It then sets the Certainty Factor (CF) to MB - MD. While MB and MD range from 0 to 1, CF ranges from -1 to 1. For simplicity, we shall discuss the MYCIN methodology in terms of its computation of MB; the same principles apply to the computation of MD.

Each MYCIN rule which provides support for a conclusion h specifies an MB[h;i] where i is the set of premises of the rule. If i is known with certainty, MB[h;i] can be used straightaway in the above computations. The difficulty arises when i itself is not known with certainty but only its CF is known. In other words, we have some evidence e on the basis of which we have computed CF[i;e], and now wish to use i as evidence in favour of h. The value MB[h;i] cannot be used as it is valid only if i is known with certainty. MYCIN uses the formula :

$MB[h;e] = MB[h;i] * \max(0, CF[i;e])$

One can interpret MB[h;i] as a measure of the reliability of the rule linking i to h. CF[i;e] is a measure of the certainty of the premises of the rule. Loosely, one can interpret the MYCIN formula as

MB(conclusion) = (Reliability of rule) * (Certainty[1] of premises)

The use of the multiplication here should be contrasted with the use of the Min operator in the fuzzy reasoning systems.

MYCIN uses formulas similar to those of fuzzy logic to deal with the logical operators **and** and **or**. If the premises of a rule consist of p_1 and p_2, then MYCIN computes

$MB[p_1 \text{ and } p_2;e] = Min(MB[p_1;e], MB[p_2;e])$

in order to compute the CF of the premises.

While Shortliffe and Buchanan(1984) disclaim any probabilistic interpretation for their model, Adams(1984) has shown that such a probabilistic basis exists for part of their model, and that this probabilistic basis enables us to understand and evaluate the model better.

Adams begins with a probabilistic interpretation of MB[h;e] :

$1 - MB[h;e] = P(e|\text{~}h)/P(e) = P(\text{~}h|e)/P(\text{~}h)$

where the second equality comes from Bayes theorem (~h denotes the negation of h). If we assume that e_1 and e_2 are independent both unconditionally and conditional on h, then we have :

$$\frac{P(\text{~}h|e_1,e_2)}{P(\text{~}h)} = \frac{P(e_1\&e_2|\text{~}h)}{P(e_1\&e_2)} = \frac{P(e_1|h)}{P(e_1)}\ \frac{P(e_2|\text{~}h)}{P(e_2)}$$

---

1. Of course, the CF of the premises could be negative; in this case Max(0,CF) yields 0 forcing MB[h;e] to 0.

where the first equality comes from Bayes theorem and the second from the independence assumptions. This is the MYCIN combine formula for MB. We can also write the formula as an updation of the probability of h; we use

$P(h) + P(\sim h) = P(h|e) + P(\sim h|e) = 1$

to get

$1 - P(h|e_1\&e_2) = [P(e_2|\sim h)/P(e_2)] * [1 - P(h|e_1)]$

In other words, a piece of evidence in favour of **h** leads to a reduction in the probability of **~h** by a factor $[P(e_2|\sim h)/P(e_2)]$.

This leads to two criticisms of the MYCIN approach :

1. Just as in the Bayesian approach, the rules must be independent. Moreover, the assumption of conditional independence is even more restrictive than the Bayesian assumption and is not consistent with the full range of values for MB (Adams, 1984).

2. The MYCIN approach of treating MB and MD separately and computing CF = MB - MD lacks any theoretical justification at all. The Adams interpretation of MD is

   $1 - MD[h;e] = P(e|h)/P(e) = P(h|e)/P(h)$

   which leads to an update formula

   $P(h|e_1\&e_2) = [P(e_2|h)/P(e_2)] * P(h|e_1)$.

   If one were to consider evidence for and evidence against **h** separately, one would get the quantities $MB[h;e_+]$ and $MD[h;e_-]$ from which one could compute the quantities $P[h|e_+]/P(h)$ and $P[h|e_-]/P(h)$. The correct way to proceed would then be to use Bayes theorem to get $P(h;e_+\&e_-)/P(h)$. The subtraction MB - MD to get CF is the most serious departure of the MYCIN system from the tenets of probability; in fact the CF does not even lie between zero and unity, but could even be negative.

Apart from the combine formula, the other crucial component of the MYCIN system is the strength of evidence or intermediate hypothesis formula

$MB[h;e] = MB[h;i] * \max(0, CF[i;e])$

Adams argues that this is not true under any reasonable assumptions. While not challenging this view, I think it is important to recognize that the MYCIN formula has a useful interpretation in terms of the well known model of Bayesian inference using uncertain data (Gettys and Wilke,1969). A straightforward manipulation of the Gettys-Wilke results leads to the formula :

$$\frac{P(h|e)}{P(h)} = \frac{P(h|i)}{P(h)} P(i|e) + \frac{P(h|\sim i)}{P(h)} P(\sim i|e)$$

Intuitively, if **i** were true, we should use the factor $P(h|i)/P(h)$, while if **i** were false we should use the factor $P(h|\sim i)/P(h)$; the Gettys-Wilke formula weights these two factors by the corresponding probabilities $P(i|e)$ and $P(\sim i|e)$. The first factor uses **i** as evidence in favour of **h** while the second uses **~i** as evidence against **h**. What the MYCIN formula does is to ignore the second term and use only the first. It is readily verified that ignoring the second term leads to the MYCIN formula. The MYCIN procedure can be justified on the ground that if **~i** is a significant piece of evidence against **h** then we should have a separate rule in the system which considers the effect of ~i. The second term in the Gettys-Wilke formula would be picked up from that rule. Thus the MYCIN formula though incorrect is not without its utility.

Adams is also quite correct that the use of the fuzzy logic operators **min** and **max** to deal with **and** and **or** require very strong restrictive assumptions. For example, probability of (**p** and **q**) is not equal to the minimum of the probabilities of **p** and **q**; if **p** and **q** are independent, the two probabilities have to be multiplied giving a probability which is strictly less than the probability of either alone. However, premises of a single rule are very unlikely to be independent; they are usually put together in a rule only because they are likely to occur together. Thus multiplication of probabilities leads to a very significant underestimate of the probability of the conjunction of all the premises of a rule. On the other hand, the MYCIN formula is likely to be an overestimate.

2.4 The Modified Mycin system

Considering all the factors discussed in the preceding sections, it was decided to adopt a slightly modified version of the MYCIN system where the certainty factors could, with some <u>caveats</u>, be interpreted as probabilities. The most significant departure from the Mycin system is the complete abandonment of the idea that CF = MB - MD. Instead, the Adams interpretation is adopted to deal with favorable and unfavorable evidence in an integrated manner.

The system adopted may be summarized in terms of the following formulas.

1. Effect of a Rule Supporting the Hypothesis

   $(1 - Cf\_new) = CF\_effective * (1 - CF\_old)$

   $CF\_effective = CF\_rule*CF\_premises$

   $CF\_premises = \min(CF\_premise1, \ldots, CF\_premise\_n)$

2 . Effect of a Rule against the Hypothesis

   $CF\_new = CF\_effective * CF\_old$

   CF_effective as earlier.

As the names suggest, CF_new and CF_old are the CFs of the hypothesis before and after applying the rule in question, CF_rule is a measure of the reliability of the rule which is part of the statement of the rule, CF_premises is the CF of the conjunction of all the premises, CF_premise1 is the CF of the first premise and so on.

This model has one difficulty which the MYCIN model does not have, viz., what value of CF_old do we use while applying the very first rule. In other words, what is the initial value of CF before any rule has been applied? In Bayesian terminology, what is the prior probability of the hypothesis? MYCIN simply sets both MB and MD to zero implying a CF of zero which (on the -1 to

+1 scale) corresponds to complete ignorance. Even a cursory exposure to Bayesian statistics is sufficient to realize that setting CF to 1/2 is totally unsatisfactory. I believe that any model of inference which wishes to be true to probability or even to ordinary human inference must have recourse to prior probabilities. I do not therefore see the need for priors as a disadvantage.

Priors can be avoided if there are no "negative" rules which provide evidence against any hypothesis, i.e., all rules are positive — supporting their conclusions. In this case, just as MYCIN simply sets MB to 0, we can set CF to 0; in the absence of negative rules, MD would be 0 and MYCIN's MB-MD would agree with our CF. Thus, this approach will give results identical to those of MYCIN in systems which have no negative rules. In many contexts, it is very natural to use only positive rules, and the pure fuzzy reasoning systems discussed in 2.2 do not permit negative rules. The possibility of reproducing MYCIN behaviour on such systems is therefore quite attractive.

The solution finally adopted was that the system would start by setting CF to zero. The rules would be applied in the order specified by the user; if the user wishes to specify prior probabilities, his very first rule should achieve an CF_effective equal to the desired prior probability, e.g., it may have no premises and have a CF_rule equal to the desired prior probability. If he does not do so, then the system would be sensitive to the relative order of the positive and negative rules; negative rules would have their greatest effect if they come last, and would be ineffective if they come first. The relative order of the positive rules <u>inter se</u>, or of the negative rules <u>inter se</u> is immaterial.

The system as implemented offers the user a choice between the fuzzy system and the modified MYCIN system as described above which we shall henceforth refer to as the MYCIN system.

3. IMPLEMENTATION IN TURBO PROLOG

3.0 Introduction

The entire discussion in this chapter assumes a good knowledge of Prolog, especially of Turbo Prolog. A rudimentary knowledge of parsers and interpreters in general is also assumed.

Traditional Prolog usually runs as an interpreter; this has meant that the Prolog program has the ability to consult the file containing the knowledge base at run time, and assert this rulebase directly into the program's source code. Prolog, in fact, has the ability to manipulate Prolog terms like any other data structure so that by using assert and retract, it can manipulate its own source code like data. With the knowledge base directly asserted into the source code, Prolog's built in backward chaining ability then does the rest of the job; the user simply calls the top level hypothesis. Providing any explanation capability is also straightforward.

Turbo Prolog, however, is a compiler; the speed that this makes possible is purchased at the cost of the loss of the above facility of traditional Prolog. In Turbo Prolog, only facts can be asserted at run time; files to be consulted at run time must also contain only facts. A fact corresponds to a Prolog rule without a body. Moreover, no free variables are allowed in the fact. (In the language of the First Order Predicate Calculus (FOPC), facts are ground atomic formulas). No knowledge base can consist only of facts; if it did, it would be a database, not a knowledge base.

This would suggest that the only solution is to include the rulebase in the source code before compilation. This, of course, has the disadvantage that if the user wants to change one of his rules, he has to edit the source code and recompile it. More importantly, this solution does not alleviate the problem of providing explanation. Any expert system must, at the very least, be able to provide a complete trace of its reasoning chain and justify each step of the chain. To do this is impossible without, in some sense, manipulating the source code as it is being executed, and storing the intermediate results. Storing even the intermediate steps is impossible in Turbo Prolog because the intermediate results would involve free variables (the variables may become fully instantiated only at the end of the inference chain; till then these variables are free). Turbo Prolog databases cannot contain free variables; hence it is not possible to use <u>assert</u> to store the intermediate results. All this means that any explanation capability required must be hard coded by the user who creates the rulebase. Effectively, this means that the user has to do everything from scratch with only the Turbo Prolog compiler to help him.

3.1 PIE — the Prolog Inference Engine

Appendix K of the Turbo Prolog Reference manual, which discusses the above problem in some detail, suggests a possible solution. This solution can be best described as a Prolog interpreter written in Turbo Prolog; a detailed description can be found in Appendix K of the Turbo Prolog manual (hereafter referred to as Appendix K), and the software itself is available on the Turbo Prolog distribution diskettes.

The heart of the system is a scanner and parser which generate a parse tree from Prolog source code. This parse tree is a data structure in Turbo Prolog which Appendix K calls the static term (<u>sterm</u>). The parse tree does not contain free variables (instead it contains the names of the variables), and predicates containing these parse trees can be <u>asserted</u> into and <u>retracted</u> from Turbo Prolog databases. An entire rulebase of Prolog statements can be read, converted into parse trees, and <u>asserted</u> into a Turbo Prolog database. To get any action out of this, however, the parse trees must be interpreted. In the case of Prolog, interpreting means that the entire <u>unification</u> and <u>backtracking</u> process of Prolog must be mimicked. This is easy to do in Prolog (Turbo Prolog) itself. The idea is to construct another tree (what Appendix K calls the active term or <u>aterm</u>) in which variables are not represented by their names as in the sterm, but by pointers to (the addresses of) actual variables; in Turbo Prolog such pointers are called reference variables. Since free reference variables can be represented by pointers to vacant addresses, reference variables are the principal means of manipulating free variables; in traditional Prolog, all variables are treated as reference variables. The interpreter while constructing the aterm must also maintain a list of variable bindings which specifies how variable names in the sterm correspond to the actual variables in the aterm. This list of bindings is maintained in a data structure which Appendix K calls the environment. Both the aterm and the environment contain free variables, and cannot, therefore, be asserted in Turbo Prolog databases; they must be passed around as arguments. The predicate unify_term(aterm,sterm,environment) is the mechanism used to convert between sterms and aterms. Like most Prolog predicates, the same predicate can convert in either direction : if sterm is given as an input, aterm can be obtained as output; if aterm is input, sterm can be got as output; if both are input, unify_term can verify whether they match each other.

The crux of the interpretation process which is done by the predicate unify_body can then be described as follows. Given a goal in the form of an sterm, split it into its subgoals (e.g., a goal might be of the form "subgoal1, subgoal2, ..., subgoal_n"), and process each subgoal in turn. If a subgoal is a predicate, convert all its arguments into aterms, and call the predicate call which is responsible for executing all predicates. The predicate call searches the rulebase for a rule whose head matches the given predicate (this matching is done by unify_term). If a match is found, a new environment is created, all the arguments are unified into that environment, and unify_body is called with the body of the matched rule and the newly created environment. The process is thus repeated recursively except when the rule has no body; in this case, the subgoal in question has been satisfied. The predicate call is also responsible for the handling of built-in predicates (if any). For these predicates, no clauses will be found in the user's rulebase; the call predicate must simply carry out the desired action.

3.2 An Expert System Shell

The idea underlying the PIE module discussed above can be extended to implement an expert system shell; after all, the major function of an expert system is to interpret the rulebase. In fact, the same idea can be used to write an interpreter for any language, not necessarily Prolog. Of course, if the input language is Prolog-like, then large parts of the PIE scanner, parser and interpreter are readily usable.

In this research, the language in which the rulebases are written is sufficiently Prolog-like to require no changes in the scanner and parser, but substantial changes are required in the interpreter. The changes arise from the need to handle uncertainty and the consequent inadequacy of straightforward backtracking as discussed below.

In normal Prolog programs, or, for that matter, in any inference system under certainty, multiple rules can be handled adequately by backtracking. The different rules will usually produce different solutions; even if they do not, the only disadvantage is that the same solution may be duplicated. Duplication of solutions can, under certainty, be easily solved by simply discarding the duplicates. But under uncertainty, this will not do : each rule produces a different piece of evidence in support of the same solution. All this evidence must be considered together to determine the certainty factor associated with the solution. In the actual implementation, therefore, the relevant clause for the predicate call0 (the difference between call and call0 is discussed later) is as follows :

```
  call0(Id,Aterml,Cf,Rule_head):-
         bind_head_unique(Id,Aterml,Concl,Rule_head),
         use_all_rules(Id,Aterml,Concl,Cf,Rule_head).
```

Here, Id is the name of the predicate to be executed, Aterml is the list of arguments, Cf is the Certainty Factor (to be computed by call0), and Rule_head is the parent goal of which the current predicate is a subgoal. Bind_head_unique returns the conclusion of the first matching rule, discarding solutions which have been found before; under backtracking, therefore, each call to bind_head_unique will produce a fresh solution. Use_all_rules then applies all the rules to obtain support for the given conclusion to compute its true certainty factor.

The predicate bind_head_unique is coded as follows :

```
bind_head_unique(Id,Aterml,
           [int(Rule_no),cmp(Id,Sterml),Sbody,
            int(Rule_cf),int(Cf_out)],
           Parent_head):-
      rule(Rule_no,cmp(Id,Aterml1),Body,Rule_cf),
      free(Env),
      unify_terml(Aterml,Aterml1,Env),
      E = Env,
      unify_terml(Aterml,Sterml1,E),
      Rule_head = [int(Rule_no),cmp(Id,Sterml1)],
      assert(why(Rule_head,Parent_head)),
      unify_body(Body,Sbody,Cf_out,Env,Rule_head),
      unify_terml(Aterml,Sterml,Env),
      unique(Rule_no,Id,Sterml).
```

The second argument of bind_head_unique (the list on the second and third lines) is the conclusion. Apart from the solution (Sterml) itself, this conclusion specifies the rule number, its CF (Rule_cf), and the CF of the premises of the rule (Cf_out). The code for this predicate is similar to that for the predicate call in Appendix K. There is some additional housekeeping being done (Rule_head and the why predicate) which will be discussed later, but the main change is the last line which discards solutions which have been processed before. This is achieved by the predicate unique coded as follows :

```
unique(Rule_no,Id,Sterml):-
      head(Rule_no,cmp(Id,Sterml)),!.

unique(Rule_no,Id,Sterml):-
      not(head(_,cmp(Id,Sterml))),
      asserta(head(Rule_no,cmp(Id,Sterml))).
```

The database predicate head stores every solution found so far along with the rule number from which the solution was obtained. The second clause for unique, checks this database to determine whether the solution is new, and if so stores this solution. If the solution has been obtained before, this clause fails causing bind_head_unique also to fail. The only exception is in the first clause for unique. If the solution has been obtained before from the same rule, unique and, therefore, bind_head_unique succeed. This situation arises when the same predicate has been evaluated before for some other purpose (several different goals might have the same or similar subgoals).

After bind_head_unique has discovered a solution, use_all_rules applies all rules to compute the certainty factor :

```
  use_all_rules(_,_,[_,Sterm,_,_,_],Cf,_):-
       theorem(Sterm,Cf),!.

  use_all_rules(Id,Aterml,Concl1,Cf,Rule_head):-
        Concl1 = [int(Rule_1),_,_,_,_],
        findall(Concl,
               use_one_rule(Id,Aterml,Rule_1,Concl,Rule_head),
               Concl_list),
        combine_all([Concl1|Concl_list],0,Cf).
```

The first clause checks the database predicate theorem which stores the certainty factor of all solutions discovered so far. Wherever possible, the system just looks up the certainty factor from this database instead of going through all the rules again. Otherwise, the second clause uses findall to gather the conclusions of all rules applicable to the given solution. This list is appended to the conclusion of the first rule, and combine_all is called to compute the certainty factor. Combine_all is as follows :

```
  combine_all([],Cf,Cf):-!.
  combine_all([
     [int(Rule_no),cmp(Id,Sterml),Sbody,int(Rule_cf),int(Cf_out)]
     |Tail],Cf0,Cf_end):-
```

```
        combine(Cf0,Cf_out,Rule_cf,Cf1,Cf),
        trace_proof(Rule_no,cmp(Id,Sterml),Sbody,
                    Rule_cf,Cf_out,Cf1,Cf),!,
        combine_all(Tail,Cf,Cf_end).
```

This recursive predicate just traverses the list of conclusions, processing each conclusion by turn. The predicate combine takes the conclusion of one rule and updates the certainty factor. All its five arguments are certainty factors; these are the current CF (before applying the rule), the CF of the premises, the CF of the rule, the effective CF from this rule alone, and the final CF after applying the rule. The nature of this combine function is discussed in 2.2 and 2.3, where the fuzzy and MYCIN methods are described. The software supports both functions.

The predicate use_one_rule is similar to bind_head_unique except that instead of testing for uniqueness, it only ensures that the first rule which has already been applied by bind_head_unique is not applied again (Rule_no >< Rule_1).

```
  use_one_rule(Id,Aterml,Rule_1,
             [int(Rule_no),cmp(Id,Sterml),Sbody,
              int(Rule_cf),int(Cf_out)],
             Parent_head):-
        rule(Rule_no,cmp(Id,Aterml1),Body,Rule_cf),
        Rule_no >< Rule_1,
        free(Env),
        unify_terml(Aterml,Aterml1,Env),
        E = Env,
        unify_terml(Aterml,Sterml1,E),
        Rule_head = [int(Rule_no),cmp(Id,Sterml1)],
        assert(why(Rule_head,Parent_head)),
        unify_body(Body,Sbody,Cf_out,Env,Rule_head),
        unify_terml(Aterml,Sterml,Env).
```

This set of predicates, together with the unify_body and other predicates as in Appendix K, constitute the core of the rulebase interpreter required for reasoning under uncertainty. Several other predicates are needed to provide the explanation capability and user interface; the more important of these are described in subsequent sections. One other point has been implicit in the above discussion of certainty factors. Predicates do not just succeed or fail; they succeed with some certainty factor or they fail. Sometimes, it is convenient to convert the failures also into successes with certainty factors of zero; ultimately, of course, it is necessary to regard a certainty factor of zero as a failure.

This solution to the uncertainty inference problem runs into difficulties with Prolog rules which are capable of producing an infinite number of solutions. For example, the following predicate if used in Turbo Prolog will produce an unending[1] list of solutions — 0, 1, 2, 3, 4, ...

```
number(0).
number(X):- number(Y), X = Y + 1.
```

This will not work in the interpreter discussed above. Its attempt to find all possible solutions and process them together will fail at some point when the system runs out of stack space.

1. This is true only in theory; actually, after several hundred solutions have been obtained, the program will terminate with a "stack overflow" error message.

3.3 Built-in Predicates

It has already been mentioned that in addition to handling the user defined predicates the system must be able to handle whatever built-in predicates the user is allowed to use. In the current system, the following built-in predicates are supported :

1. Arithmetic. Only integer arithmetic is supported. The Prolog predicate <u>is</u> does arithmetic (similar to = in Turbo Prolog) with the usual arithmetic operators. Some basic functions like <u>abs</u>, <u>max</u>, <u>min</u>, <u>mod</u> are supported. The operators <u>and</u> and <u>or</u> are implemented as arithmetic operators to facilitate fuzzy logic applications; they are internally converted into <u>min</u> and <u>max</u> respectively. A predicate <u>pct</u> is provided for computing percentages without producing errors like division by zero and integer overflow. A function <u>ramp</u> is provided for applying fuzzy logic; its definition is as follows (its application is described in 4.1):

   ramp(Value,Start,End) =

ramp_up(Value,Start,End)	if Start $\leq$ End
100 − ramp_up(Value,End,Start)	if Start > End

   ramp_up(Value,Start,End) =

100	if Value $\geq$ End
0	if Value $\leq$ Start
(Value−Start)*100.0/(End − Start)	otherwise

2. Relational operators < and >.
3. Manipulating certainty factors. The predicate <u>cf</u> takes a

proposition as its argument, computes its certainty factor, and returns this as a numerical value. The predicate fuzzy takes a numerical value and succeeds with that value as its certainty factor. These two conversion predicates allow the user to override the standard combine functions and apply any method whatsoever for computing certainty factors.

4. Input/Output . Knowledge bases are not supposed to do any input or output; that is the sole prerogative of the control program. Input/output facilities are, however, useful for questions and for sophisticated debugging. The predicate <u>read</u> reads a line from the terminal, parses it into an <u>sterm</u>, and then converts that into an <u>aterm</u>; in effect an arbitrary Prolog term (<u>aterm</u>) can be read from the terminal. The predicate <u>write</u> writes an arbitrary list of Prolog terms (<u>aterm</u> or <u>sterm</u>) in a readable manner.
5. Questions. The predicates — bound, not_asserted, assert,— are intended to be used in questions; their function is discussed in 3.4 below.

The predicate <u>Call0</u> which is responsible for the execution of all predicates contains separate clauses for each of the built-in predicates. More built-in predicates can be provided simply by adding more clauses to <u>call0</u>. For example, the clause for the relational operator "<" is as follows :

```
 call0("<",[T1,T2],100,_):-!,
        bound(T1), bound(T2), eval(T1,R1,cmp("<",[T1,T2])),
        eval(T2,R2,cmp("<",[T1,T2])),R1 < R2.
```

The rule first checks that the two terms T1 and T2 which are being compared are both bound. Then the two terms are evaluated to obtain the numerical values R1 and R2 using the predicate <u>eval</u>. Finally R1 and R2 are compared using Turbo Prolog's built-in relational comparison operator. The predicate <u>eval</u> takes three arguments — the term to be evaluated, the result of the evaluation, and the goal for which this evaluation is being done. The last is part of the housekeeping required for providing the explanation capability as discussed later.

3.4 The Rulebase

The rulebase to be created by the user consists of rules and questions. Rules are written as follows :

```
rule(rule_no,rule_text,rule_cf)
```

Rule_no is an integer which gives a number to each rule. Rule_text is a string which must be enclosed in quotation marks. If the rule_text contains embedded quotation marks, the single quotation mark should be used; the double quotation marks are reserved for the marks enclosing rule_text. Otherwise, rule_text is an ordinary Prolog rule; the only difference between this and ordinary Prolog is that the only standard predicates supported are the ones listed in 3.3 above. In particular, the predicates cut (!) and fail are not supported. No genuine rulebases should have these predicates. Rule_cf is an integer giving the certainty factor of the rule in percent. Rules against a hypothesis (negative rules, see 2.4) are entered with negative certainty factors. One change which has been made from conventional Prolog is that the word "if" can be used instead of the Prolog symbol ":-" for better readability.

Questions are entered as follows

```
question(question_number,question_text,question_cf)
```

These arguments are identical to those for rule. In particular, question_text is an ordinary Prolog rule. The difference is only in the way the system processes it. A question is intended to be user where all rules have failed to produce a positive or negative answer, and it is necessary to obtain an answer before the inference can proceed; the question, therefore, asks the user to give the answer. The question will, therefore, be used by the system only if all rules have failed. For example, the cost variance investigation system requires the variance data to be available for any inference to be carried out. What happens if the user queries the system about a variance for which data does not exist? One possibility is that the system should prompt the user to enter the variance data, accept this data and assert it into the database. However, if the user asks for a list of all variances, the question should not be invoked; the system should only give those variances for which data exist. This is why the question cannot be treated like a rule.

The predicate <u>call</u> is used in conjunction with <u>call0</u> to handle rules and questions in the manner suggested above :

```
call(Id,Terml,Cf,Parent):-call0(Id,Terml,Cf,Parent).

call(Id,Aterml,Cf,Parent_head):-
       question(Q_no,cmp(Id,Aterml1),Body,Q_cf),
       free(Env),
       unify_terml(Aterml,Aterml1,Env),
       not(call0(Id,Aterml,_,Parent_head)),
       E = Env,
       unify_terml(Aterml,Sterml0,E),
       Rule_no = -Q_no,
       Rule_head = [int(Rule_no),cmp(Id,Sterml0)],
       assert(why(Rule_head,Parent_head)),
       unify_body(Body,Sbody,Cf_out,Env,Rule_head),
       unify_terml(Aterml,Sterml,Env),
       combine_all([  [int(Rule_no),cmp(Id,Sterml),
                      Sbody,int(Q_cf),int(Cf_out)]
                   ], 0, Cf),
       asserta(assertion([int(Cf),cmp(Id,Sterml)])).
```

The first clause for call simply calls the predicate call0 discussed earlier. All rules will, therefore, be applied. If no rules succeed, or when all solutions have been exhausted, call0 will fail. Now Prolog automatically backtracks, and tries the second clause for call. This clause first checks whether any question matches (these three lines are similar to the lines of call0 for processing rules). The next line

not(call0(Id,Aterml,_,Parent_head)),

is the crucial line which ensures that no rule has succeeded even with a certainty factor of zero. The rest of the code for call0 is similar to the code in call0 which deals with rules; the processing of rules and questions is very similar. The only addition is the last line which stores the conclusion of the question in the database of assertions so that the same question will not be asked of the user again.

The question is like any other rule and can carry out any amount of reasoning and input-output. The user input can be used to obtain values for the variables used in the question. At the end of it all, the system records the conclusion with a certainty factor equal to the certainty factor with which the question itself succeeded. The question can use the built-in predicate <u>fuzzy</u> to succeed with a user supplied certainty factor.

All input and output must be done by <u>read</u> and <u>write</u>; the system itself does not automatically do any input or output.

Whenever a <u>read</u> is being executed, the user can type <u>why</u> instead of giving an answer; the system responds with what goal it is attempting; answering <u>why</u> again will reveal the supergoal for which this goal is being attempted, and so on.

Finer control over the execution of questions is possible by using the built-in predicates <u>bound</u> which tells whether a variable is bound, <u>not_asserted</u> which consults the data base of assertions discussed above, and <u>assert</u> which allows an arbitrary fact to be asserted into the assertions database. Using this it is possible to specify more precisely the conditions under which the question is to be processed further. The predicate <u>assert</u> allows a question to obtain data about a number of things and assert all of them into the database. In fact, the question itself can fail so that its conclusion is not recorded. Using these predicates it is even possible to obtain user input using rules rather than questions.

The predicate <u>call0</u> has an additional clause :

```
  call0(Id,Aterml,Cf,_):-
        assertion([int(Cf),cmp(Id,Sterml)]),
        free(E),
        unify_terml(Aterml,Sterml,E).
```

which consults the database of assertions. An assertion made by the user is thus available for the rest of the session.

3.5 Data

Strictly, all data can be entered as part of the rulebase itself, but this is excruciatingly slow, particularly if computations have to be done to transform the data into more usable form. In the case of cost variance investigation, the data might consist of actuals and budgeted figures; these have to be converted into variance amounts and percentages. If the data were included in the rulebase along with the rules for converting the data into variances, the system slows down as all these rules have to be interpreted again and again. This, of course, brings us back to the advantages of compilers discussed in 3.0. A solution is to hard code these computations into the source code of the control program itself, and treat <u>variance</u> as a built-in predicate which the system interprets as a request to consult the internal database. Any changes in this predicate requires changing the source code and recompiling.

The built-in predicate variance is interpreted as follows :

```
   call0("variance",
         [atom(Item),atom(Div),int(M),int(Q),int(Mp),int(Qp)],
         100,_):-
       variance(Item1,Div1,M1,Q1,Mp1,Qp1),
       Item=Item1,Div=Div1,M=M1,Q=Q1,Mp=Mp1,Qp=Qp1.
```

Processing the datafile consists of consulting it to create the database data and then calling the following predicate which performs the computations and creates the variance database.

```
  compute_variances:-
        data(Item,Div,Mbud,Mact,Qbud,Qact),
        Mvar = Mact - Mbud,
        Qvar = Qact - Qbud,
        pct(Mvar,Mbud,Mpct),
        pct(Qvar,Qbud,Qpct),
        assert(variance(Item,Div,Mvar,Qvar,Mpct,Qpct)),
        fail.
  compute_variances.
```

This particular set of computations is for the case discussed in Chapter 4. The data in that case consists of the item (e.g. sales, profits etc.), the division, monthly budgets and actuals, and quarterly budgets and actuals. The variance database computed therefrom by the above predicate contains the item, division, monthly and quarterly variances (amounts), and monthly and quarterly percent variances. The predicate pct is used to ensure that division by zero leads to a large number (±10000%) rather than an integer overflow.

3.6 Explanation Capability

Explanation consists essentially of three things :

1. Which rules succeeded and why?
2. Which rules failed and why?

3. Why was a rule tried?

To do these a complete trace of the inference process has to be maintained. Whenever a rule succeeds, the rule head and the rul body are stored along with the rule number and the various certainty factors in a predicate called rule_fired. The rule head and rule body are the instantiated terms in which all variables have been replaced by their values. To justify any conclusion, the system consults this database to find out which rules succeeded. For each such rule, the rule body is looked up to find out the premises involved. Each of these is recursively justified in turn to any required depth of justification.

To explain complex arithmetic calculations, the evaluation predicates store intermediate results of all computations in the form – parent_goal, expression, value. The parent_goal is stored so that when justifying that goal, this database can be looked up to explain the arithmetic behind it.

Whenever unify_body fails, the failure is recorded in the form – failed_clause (literal), failed clause (instantiated), parent_goal, reason. In the literal clause, all variables are represented by their names; this is the form in which the clause appears in the rulebase; in the instantiated clause, all variable which are bound at the time of failure are replaced by their values. There could be several reasons for a failure :

1. no rules may have matched
2. some rules may have matched, but all of them failed

3. the rules might have succeeded with zero cf.

4. the clause consists of a negative - not(negated_clause
   and the negated_clause succeeded

5. the clause consists of one of the built-in predicates
   is, =, <, >, and the predicate failed.

Because of the backtracking that goes on, it is not quite straightforward to ascertain the reason for the failure. Some reasoning is required to figure out this out correctly, but Prolog is quite well suited to carry out such reasoning.

In case (2), a complete explanation of the failure must explain the failure of each of the rules. Using the internal database <u>failed</u> this can be done recursively to any depth required.

Whenever a subgoal is created, a database entry is made in the <u>why</u> predicate which contains both the new subgoal and the parent goal of which it is the subgoal. This allows unlimited backward chaining to explain why a certain subgoal was or is being attempted. When a user response is being asked for in response to the built-in predicate <u>read</u>, the latter situation obtains : the goal is being attempted, and has neither succeeded nor failed. This is why the other database on failed and succeeded clauses cannot be relied on to do this backward chaining from subgoal to parent goal.

3.7 User Interface

The user interface consists of two windows :

1. The main goal and dialog window in which the user types in goals (queries) and the system provides answers and justification.
2. The command window in which the user types in various commands for changing settings of various internal parameters or requests explanations, or issues other commands.

The system toggles between the two windows with the ESC key; pressing ESC in the goal window immediately brings up the command window and vice versa. The command window is only for accepting the user's command; as soon as the command is typed in, the command window disappears and the system's response usually appears in the dialog window. The main purpose of the command window is to clarify to both the user and the system that the user's input is to be treated as a command and not as a goal. Some commands cause a new window to be opened up. For example, the edit command opens up a window in which the rulebase is displayed for editing. The view command, sorts all solutions by their certainty factors and displays them in a window in which the user can scroll through them at leisure. The help command displays a help file.

All input from the user follows a Prolog-like syntax. This is because the same parser used for parsing the rulebase is used to parse the commands before executing them. In fact, the command processor can be viewed as an interpreter which interprets a Prolog-like language of commands.

3.8 System Requirements, Performance and Enhancements

The system is quite large - the executable file is about 200k, and since Prolog requires heaps of memory (for the heap!), a minimum of about 512k of free memory is required to run. Large rulebases require correspondingly larger amounts of memory. A PC with no resident programs gives enough memory, but I find the system running short of memory on my system unless I remove most of my resident programs. Moreover, Turbo Prolog's garbage collection algorithms appear to be less than perfect. After repeated editing of the rulebase with the associated retracts and asserts, the internal heap memory apparently becomes excessively fragmented, and the system runs out of memory. It is necessary to start afresh; but since the files are saved, nothing is really lost. The garbage collection process is also quite slow; when the user types in a new goal after evaluating a complex goal, a lot of time is spent in retracting all the old proof traces and the associated garbage collection before the new goal is processed; all that the system can do is to display an appropriate message requesting the user to wait.

The system obviously does not run from within the Turbo Prolog environment; it is necessary to compile it to an .EXE file. Since the program exceeds 4000 paras of code, this makes it necessary to split it up into three modules each of which is less than 4000 paras and compile them as a project. Otherwise, the calls within each module would become "far" calls slowing down the system and consuming even more memory.

All this means that the system is close to the limit of what is possible in the PC environment with Turbo Prolog. Any further enhancements will be at the cost of the heap memory available for the system to run.

In terms of speed, the system is quite fast on small rulebases. But, on large rulebases, the system response does become somewhat sluggish.

The main enhancement that one would like to add to the system is a rudimentary natural language facility. At the very least, the system's output should be more English-like rather than Prolog-like. This is not too difficult because all the output comes from Prolog structures that are essentially parse trees. There are hardly any unparsed strings being simply written out. To convert a parse tree into an English-like output is by no means a daunting task. The principal consideration which deters such an enterprise is the fact that, as discussed above, the system has very little spare memory to accommodate this additional code.

Handling English-like input is a greater problem as parsing English is far more difficult than parsing Prolog. The time honored solution in AI of using a simple minded parser and simply asking the user to clarify any residual ambiguity makes the task less difficult. With the powerful parser generators now available, providing a very rudimentary English-like input facility is not an insuperable obstacle if enough memory and processing power are available.

The above considerations lead to the conclusion that further work in this area would have to move out of the PC environment to the workstations or to other more powerful computing platforms.

4. BUILDING THE KNOWLEDGE BASE : AN ARMCHAIR CASE STUDY

4.0 The Modo Company

A case on variance investigation from a well known book on Management Control Systems (Antony, Dearden and Bedford, 1984) was chosen for an armchair case study to demonstrate the building of a knowledge base for cost variance investigation.

The Modo Company case (see Appendix One) consists mainly of a set of variance reports circulated to the directors of the Modo Company prior to the board meeting. The protagonist in the case is a director who has to decide what questions to ask at the board meeting.

The case provides variance data on divisions labeled A to R. The variances covered are – income, sales, receivables and inventory.

A perusal of the case reveals some exceptional situations which an Expert System cannot be reasonably expected to handle. These must be handled manually while providing the data to the system. For example, division J was apparently to be split off from H this year, but this has not happened. The budgeted figures for J and H must therefore be combined.

4.1 Formulating the Traditional Rules

Traditionally accountants have used a variety of simple rules to filter out insignificant variances. The most important of these rules are

1. Examine all variances above a specified rupee or dollar amount.
2. Examine all variances which are more than a specified percentage of the budget.

This set of rules is very popular, and appeals to our common sense. It can also be justified in terms of statistical decision theory. A variance of the first kind is important because the potential benefit from investigating it is high. A variance of the second kind is important because it indicates a high probability that something is wrong.

Another commonly used set of rules relate to the use of past data. Often variance for the current month are reported along with variances for the whole budget period or year to date (YTD). In this case, a relatively small monthly variance coupled with a significant YTD variance implies that the monthly variance is worth investigating as it is a continuation of a bad trend. Again, this rule has a sound statistical foundation which can be elaborated in terms of the CUSUM (cumulative sum) test in control chart theory.

These two sets of rules (particularly the first set) are often applied mechanically, and only the "significant" variances are even reported to the senior managers. Beer (1973) has argued strongly for the mechanical application of highly sophisticated statistical rules of the second type; he has also reported his experience with the application of such rules in Allende's Chile.

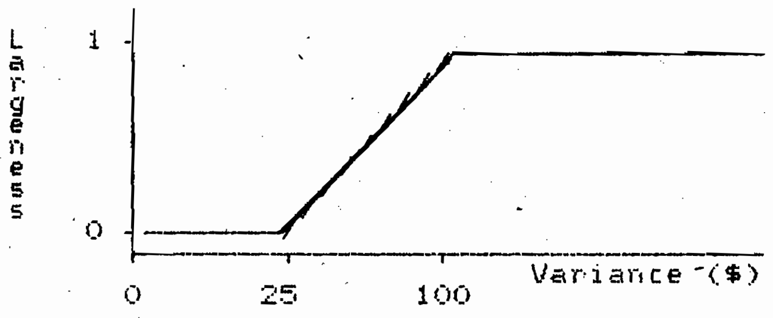
Other rules are usually applied subconsciously if at all. For example, managers in a particular company may know that in their context, a variance in sales volume is more important than a variance in production costs even if both have the same rupee impact on final profits. This kind of knowledge is reflected in actual decision making mainly by the manner in which variance aggregation is done.

First of all, it is necessary to build into the model the knowledge corresponding to the commonly used traditional rules discussed above. The rules involved are reasonably clear and straightforward; the major novelty is the handling of uncertainty and fuzziness. The traditional rule might say that all income variances above \$50[1] or above 10% are to be investigated. The intention, however, is to investigate large variances, and the limits specified above are only a rough and ready operational definition of largeness. The rule as formulated for the expert system would recognize that the concept of large is in fact a fuzzy one; there is no magic number at which a variance becomes

---

1. All dollar figures in this chapter and in appendices Two and Three are in thousands; so we are really talking of a limit of \$50,000.

large. One might, for example, be sure that any income variance below $25 is small in absolute terms, and anything above $100 is definitely large in absolute terms. Similarly one may be sure that 5% is definitely small and 20% definitely large in relative terms. One might not be very sure as regards anything in between. The use of fuzzy logic allows one to retain this inexact notion of largeness, and let the expert system reflect that uncertainty in its inference procedure. In fuzzy logic, one is not forced to classify all variances as large or not large, but is allowed to distinguish several grades of largeness by attaching a number between 0 and 1, where 0 represents definitely not large and 1 represents definitely large. For simplicity, the ramp shaped function has been used for this purpose. This is graphed below for the case of the income variance ($ amount) :



In the input to the expert system, this function is represented as :

```
ramp(variance_amount,25,100)
```

which indicates that the ramp starts from 0 at 25 and reaches 1 at 100.

Similar ramps are defined for percent variance. The definition used for income variances is :

```
ramp(variance_percent,5,20)
```

which indicates that a variance of 5% or less is definitely small, and that a variance exceeding 20% is definitely large.

The usual notion that a variance is large if it is large in absolute amount or in percent terms would be represented in the expert system by writing :

```
ramp(variance_amount,25,100) or ramp(variance_percent,5,20)
```

As explained in 2.2, in fuzzy logic, the logical operator <u>or</u> is implemented by the arithmetic operator <u>max</u>. Internally, therefore, the system would actually compute :

```
max(ramp(variance_amount,25,100), ramp(variance_percent,5,20))
```

It is now possible to formulate the first rule for income variances; informally, the rule states that a variance must <u>certainly</u> be probed if the variance is large where the fuzzy notion of largeness is formalized as above. The rule should, therefore, behave as follows : if the variance exceeds $100 or

20%, then the variance must definitely be probed; its certainty factor is equal to 100%. If the variance is below $25 and below 5%, then the rule provides no evidence for probing the variance; the certainty factor remains at 0. Even in this case, the variance is not completely disregarded; some other rule may provide evidence for probing the variance. If the variance is in neither of these two categories, the certainty factor would be somewhere between 0 and 1; subsequent rules may then change this value in the light of other evidence.

The rule as it appears in Appendix Two is as follows :

```
rule(10,"
    probe(income,Div) if
        variance(income,Div,Mvar,Qvar,Mpct,Qpct),
        Mvar < 0,
        fuzzy(ramp(-Mvar,25,100) or ramp(-Mpct,5,20)).
    ",100). /* large negative monthly variance*/
```

The first line identifies the rule as rule no.10; the last line states that the certainty factor of the rule (the reliability of the rule itself) is 100%; the text between /* and */ is a comment. The second line says that the income variance of any division - Div - is to be probed if the premises below are satisfied (in Prolog, all words beginning with a capital letter are variables; Div is a variable and the rule can be applied to any of the divisions of the Modo company). The very next line consults the variance data. This data is stored in the predicate variance in which the first argument is the type of variance (income or sales), the second is the name of the division and the remaining are the variance figures - Mvar is the monthly variance amount, Mpct is the monthly variance percent, Qvar and Qpct are the variance amount and percent for the quarter. Variance is always computed as actual minus budget; depending on the nature of the item, a positive variance may be favorable or unfavorable.

The next line "Mvar < 0" states that the rule does not apply to positive (favorable variances). The final premise of the rule uses the definition of large as discussed above, and then uses the predicate fuzzy to convert this numerical value into a certainty factor. The fuzzy predicate succeeds with a certainty factor equal to the numerical value of its argument. The certainty factor coming out of the application of the rule is also equal to this numerical value because the rule has a certainty factor of 1.

The above rule dealt with the largeness of the monthly variance; the next one handles the quarterly variance. The approach adopted here is that a large quarterly unfavorable variance is an indication that an unfavorable monthly variance is to be investigated even if it is of only moderate size. The rule for this would require a premise of the form "monthly variance is at least moderate AND quarterly variance is large". After defining the appropriate ramps, the rule looks like this :

```
rule(20,"
    probe(income,Div) if
        variance(income,Div,Mvar,Qvar,Mpct,Qpct),
        Qvar < 0, Mvar < 0,
        Moderate is ramp(-Mvar,10,50) or ramp(-Mpct,3,10),
        Large is ramp(-Qvar,50,200) or ramp(-Qpct,5,20),
        fuzzy(Large and Moderate).
    ",50). /* large quarterly with moderate monthly variance*/
```

This rule is regarded as less strong than the earlier one and has a rule certainty factor of only 50%.

Rules similar to the above are used for sales variances and the accounts receivable and inventory variances. In the case of the last two, the variance is computed from a "flexible" budget which adjusts for deviations of sales from budget. In the absence of data, it is assumed that the budgetary norms for number of days sales in receivables and for inventory turnover are the same as the last year's actuals. Under this assumption, it is possible to construct a flexible budget from the given data and compute the dollar and percent variances from this budget.

4.2 Analytical Rules

This section looks at some analytical rules which rely on management accounting theory.

Management accounting theory would break up an income variance into several components. Some of these may be the contribution lost due to lower sales volume, the variance in contribution margin (due either to lower unit price realization or higher variable costs) and variances in fixed costs. The manager might be more concerned about some of these variances than about others.

Data does not exist in the case to carry out all this variance analysis. A casual look at the data reveals, however, that some divisions have failed to reach budgeted profits despite having sales well in excess of budget; this very clearly indicates a substantial drop in the contribution margin. Prima facie, this deserves investigation even if the income variance itself is not large.

This analysis leads to a rule which says that an income variance is to be investigated regardless of its magnitude if there is any evidence for a margin squeeze. Unfortunately, contribution data is not available for the computation of the C/S (contribution/sales) ratio. But one can proceed as follows :

$$\begin{aligned}\text{actual income} = {} & (\text{budgeted C/S})*(\text{actual sales}) \\ & + (\text{actual C/S} - \text{budgeted C/S})*(\text{actual sales}) \\ & + (\text{variance in fixed costs})\end{aligned}$$

If the third term is ignored or subsumed along with the second under the term "margin variance", one can write :

$$\begin{aligned}\text{income variance} = {} & (\text{budgeted C/S})*(\text{sales variance}) \\ & + (\text{margin variance}).\end{aligned}$$

If sales variance is favorable, then

$$\text{margin variance} < 0 \quad \text{iff} \quad \frac{\text{income variance}}{\text{sales variance}} < \text{budgeted C/S}$$

If sales variance is unfavorable, then

$$\text{margin variance} < 0 \quad \text{iff} \quad \frac{\text{income variance}}{\text{sales variance}} > \text{budgeted C/S}$$

The only thing in these inequalities that can be computed from the given data is the ratio : (income variance)/(sales variance). However, if this ratio has a very low or negative value, one can be reasonably sure that it is below the budgeted C/S ratio, and if it is very high, it is likely that it exceeds the budgeted C/S ratio. This leads to the rules :

1. There is a margin squeeze if sales variance is unfavorable and the ratio (income variance)/(sales variance) is very high.
2. There is a margin squeeze if sales variance is favorable and the ratio (income variance)/(sales variance) is very low or negative.

The terms very high and very low are, of course, treated as fuzzy concepts, and operationalized by appropriate ramp functions. The rule which says that an income variance is to be probed if there is any evidence for a margin squeeze is given a certainty factor of 50%. The rules are formulated as follows :

```
rule(30,"
    probe(income,Div) if
        variance(income,Div,Delta_profit,_,_,_),
        variance(sales,Div,Delta_sales,_,_,_),
        Ratio is pct(Delta_profit,Delta_sales),
        margin_squeeze(Ratio,Delta_sales).
    ",100).  /* margin squeeze in monthly data */
```

```
rule(110,"
     margin_squeeze(Ratio,Delta_sales) if
          Delta_sales > 0,
          fuzzy(ramp(Ratio,5,-30)).
     ",100). /* incremental sales have not earned
                 even 5% contribution */

rule(115,"
     margin_squeeze(Ratio,Delta_sales) if
          Delta_sales < 0,
          fuzzy(ramp(Ratio,25,80)).
     ",100). /* contribution lost exceeds 25% of sales drop */
```

Another rule which is derived from management accounting theory is that a sales variance is more disturbing if it is accompanied by a lengthening of the average collection period. The implication in such a case is that sales are falling despite the more liberal credit policy. The rule required to deal with this is quite simple :

```
rule(530,"
     probe(sales,Div) if
          variance(sales,Div,Mvar,Mpct,_,_),
          Mvar < 0,
          wc_variance(ar,Div,_,Pct),
          fuzzy(ramp(Pct,5,20) and
                (ramp(-Mvar,50,300) or ramp(-Mpct,3,10))).
     ",50).  /* negative sales variance with lengthening ACP*/
```

The working capital variances are stored in a predicate of the form wc_variance(Item,Division,Amount,Percent); in this case monthly and quarterly data are not separately available.

4.3 Overall Divisional Performance

The above rules have discussed the decision to probe individual variances. An equally interesting question is that of assessing the overall performance of the division.

The simplest rule would be that the division needs to be looked into if any of its variances are large. A more refined approach would recognize that not all variances are equally important. An income variance is the most important with sales variance only slightly less important. The working capital variances are much less important. This assessment of relative importance is incorporated in the certainty factors of the corresponding rules which are listed below :

```
rule(1,"
     probe(Div) if
          probe(income,Div).
     ",100).  /* Probe division if income variance significant*/

rule(2,"
     probe(Div) if
          probe(sales,Div).
     ",100).  /* Probe division if sales variance significant*/

rule(3,"
     probe(Div) if
          probe(ar,Div).
     ",50).  /* Probe division if A/R variance significant*/

rule(4,"
     probe(Div) if
          probe(invy,Div).
     ",50).  /* Probe division if inventory variance significant*/
```

4.4 Exceptions

Finally, this section gives an illustration of a rule which handles exceptional cases. This rule is also exceptional in the sense of providing evidence against the hypothesis rather than in favour of it.

The exception involved is that of Division C which is apparently being closed down. In this case, one would expect some significant variances from budget which are not to be taken seriously. The rule is very simple :

```
rule(1000,"
    probe(_,c).
    ",-80).  /* ignore Division c variances (being closed) */
```

It simply says that the evidence for probing a variance in division C is to be ignored by reducing its certainty factor by 80%. This reduces the certainty factor to 20% or less which is below the threshold certainty factor. It still keeps the variance alive so that strong variances in various items in division C could collectively add up to a moderate certainty factor for probing the division as such. If one were absolutely sure that one did not want to look at division C at all, one could change the certainty factor of the above rule to -100% which would completely kill all evidence for probing any variance. (This effect could, perhaps, be more simply achieved by deleting the data for C from the input data itself).

4.5 Performance of the Expert System

Appendix Three is a log of a sample session with the system; it gives the output of the system in response to several different types of queries. In this appendix all text entered by the user is underlined; all comments (between /* and */) are shown in **boldface**; everything else is the system's response.

The session begins with the query "probe(income,_)", i.e., which income variances are worth investigating. The system responds with a list of 7 variances and also reports that there were two others whose certainty factors were negligible (below the threshold value of 20%). The system had been configured (through the configuration file) to provide only brief traces of the proofs. This means that only the first level justification of the conclusion is provided; this justification consists of listing the rules which succeeded, and the computations of the certainty factors. The certainty factors of the rule and of the premises are used in conjunction to compute the effective CF yielded by the rule in isolation; this effective CF is used to update the old CF to arrive at the cumulative CF so far.

The output of the system exceeds a screenful and a good deal of it scrolls out of view. The user, therefore, issues the view command to view the solutions at leisure, scrolling up and down as he pleases. Moreover, under the view command the solutions appear sorted according to their certainty factors which makes it easy to see which are the really important ones and which are only borderline cases.

Having looked at this in detail, the user wants more detailed justifications of some items. The user recalls that division D's income variance was almost negligible, and is surprised to see a moderate certainty factor associated with it. He, therefore, issues the command trace_all to obtain a detailed trace, and then asks the system to justify this conclusion. The system does not need to perform any fresh inferences; it simply justifies its earlier conclusion in a more detailed fashion. Now the system lists each of the premises of the rule, and justifies each of them in turn to as deep a level as necessary.

The user is satisfied with this explanation; if he were not, he would probably edit the rulebase and then analyze the problem again. He now turns to another conclusion of the system which puzzled him. He wanted to know why L's income variance which seemed rather high to him was not receiving a certainty factor higher than the 65% that the system assigned to it. The command "refute probe(income,l))" brings forth an explanation of which other rules were tried and why they failed. For each failing rule, the clause that failed is pinpointed; where necessary the reasons for the failure is in turn explained.

Satisfied on this score, the user moves on to his next query as to whether division O needs to be investigated. The system concludes yes with a certainty factor of 65%. The system had earlier been asked to trace in full; it, therefore, provides a complete justification of its answer. Rules 1, 3 and 4 corresponding to the income, A/R and inventory variances were invoked. The proof trace shows that the principal reason for probing the division is its large inventory variance, and that the other variances make only small contributions to the decision to probe the division.

Before moving on to the next question, the user decides to restore brief tracing; detailed tracing produces voluminous output which clutters up the screen. Subsequent queries ask for listing all the sales variances to be investigated and all the divisions to be probed.

In substantive terms, the system's performance appears to me to be quite satisfactory. Each reader can arrive at his own judgment on this matter by comparing his own analysis of the Modo company case with the output of the system as given in Appendix Three.

More important than the success or failure of the specific rulebase developed here for a specific case is the question as to whether the methodology is versatile enough to be extended and modified to handle real life situations.

Questions also arise as to the adequacy of the user interface in terms of

1. Formulation of rules.

2. Formulation of queries.

3. Other commands.

4. System response.

These important questions are addressed in the next chapter.

5. CONCLUSIONS

5.0 Performance of the System at Current Stage of Development

The substantive performance of the system in the armchair case study discussed in the preceding chapter is quite encouraging. The system is able to integrate rules drawn from management accounting theory with statistical considerations as well as personal beliefs and preferences of the user in a straightforward manner to yield conclusions which are intuitively reasonable. On problems of this level of complexity and size, the system's speed is satisfactory, and its memory requirements are manageable. However in terms of speed and space, the system is close to the limits of what is possible in the PC environment with Turbo Prolog. It is likely that further work in this area will have to move out of the PCs to the workstations or to other more powerful computing platforms.

The log of the sample session in the previous chapter demonstrates that the system's inference capabilities and its understanding of its own knowledge base are strong enough to provide reasonable answers to a variety of questions; this illustrates the power of the expert system/ artificial intelligence methodology.

The system's Prolog-like user interface is quite acceptable in research and classroom settings. This would, however, be a major handicap in actual applications. The system is unable to provide English-like answers or to understand English-like queries, let alone accept English-like rules. The rules, queries and answers all follow a Prolog-like syntax. Since Prolog is essentially the language of the First Order Predicate Calculus (FOPC), the interface is quite powerful and versatile; as I have found to my delight, the Prolog-like interface is very convenient to the trained user. The point is that this may not be acceptable to the average user. Having argued at the outset that it is unrealistic to expect all managers to learn accounting theory and statistical theory, I am not about to suggest that they should all learn Prolog instead. A natural language interface is extremely important; no amount of menu driven user interfaces and other devices fashionable in certain so-called user friendly software packages is an adequate substitute for this.

As stated earlier (see 3.8), the programming effort required for providing a more English-like output is quite small; the only difficulty is that as enhancements are made, the system starts running out of memory and becomes too slow. The principal difficulty in providing a rudimentary English-like input facility is also one of speed and memory limitations of the PC rather than the programming effort per se.

I doubt whether it is equally easy to permit rules to be formulated in an English-like syntax, though it may be possible to have a more English-like vocabulary. (The system already permits the use of the word "if" instead of the Prolog symbol ":-"; this is an example of a vocabulary change which leaves the syntax almost unchanged). I am of the opinion that the formulation of rules will require the full power of a formal logical language, and I am not at all enthusiastic about trying to dilute this by permitting an English-like syntax.

5.1 Research Issues

The current study started out with two main research issues:

1. How can the "knowledge and experience" of the managers be formalized, generalized and shared within the organization?
2. How can the theoretical insights and heuristics suggested by accounting theory, statistical analysis and mathematical modeling be integrated with the knowledge of the managers as so formalized?

I believe that the experience with the current system has shown that the first research problem is solvable within the framework of expert system methodology. A knowledge base can indeed be built up incrementally to incorporate a good deal of background knowledge scattered throughout the organization.

The second research problem is also partly solvable. A number of heuristics derived from theoretical considerations can be readily incorporated in the knowledge bases. The difficulty that remains is one of integrating statistical theory with the fuzzy logic and certainty factors of the inference system under uncertainty. The simple fact of the matter is that the devices used in these reasoning systems do not conform to the classical laws of probability, and that the Bayesian system which does so conform is impractical in the current context. Though there has been some research on hybrid random variables which besides being fuzzy are random in the standard statistical sense, this research is not yet readily applicable to the problem of cost variance investigation.

As argued in the initial chapter itself, the problem of cost variance investigation is conceptually a problem in statistical decision theory. This theory essentially boils down to the computation of several crmplex conditional expectations; but the data for such a computation does not exist. (If the data did exist, one should be developing a numerical integration software rather than an expert system). Many of the rules in the knowledge base can be thought of as providing some evidence relating to this unknown data; for example some of the rules might suggest that the integrand is likely to be large, or that the probability density with respect to which one is integrating is concentrated in a certain interval. This data is of course available in the form of fuzzy values or certainty factors and not in the form of a point estimate or a statistical distribution. Is it possible to make use of this data to compute the integrals involved even approximately, or to make any inferences about them of either a statistical or a fuzzy nature? I am not aware of any theory which makes this possible. In the absence of such a theory, one is forced to rely only on a broad qualitative picture of the problem to convert the fuzzy knowledge of the parameters into certainty factors for the investigate/no-investigate decision. The difficulty is that the decision to investigate or not depends in a highly nonlinear manner on the various parameters involved. The rules employed by the expert system are bound to be gross oversimplifications of the theoretically correct ones. Though the system may give intuitively reasonable answers, one is left with the feeling that this is not enough. It is not enough to mimic a so-called expert; the real task is to surpass him by a substantial margin. In this respect, the expert system methodology appears to be at a dead end; perhaps real progress in this area will come from purely statistical approaches to the problem.

REFERENCES

Adams,J.B. (1984), "Probabiistic Reasoning and Certainty Factors" in Buchanan and Shortliffe(1984).

Antony, R.N., Dearden, J. and Bedford, N.M. (1989), Management Control Systems, Homewood, Illinois, Richard Irwin.

Avenhaus, R and Frick, M. (1977), "Game Theoretic Treatment of Material Accountability Problems" Part I", International Journal of Game Theory, 5, 117-135.

Beer, S. (1973), Fanfare for Effective Freedom; Cybernetic Praxis in Government, Third Richard Goodman Memorial Lecture delivered at Brighton Polytechnic, Moulsecoomb, Brighton.

Buchanan, B.G. and Shortliffe, E.H. (1984), Rule Based Expert Systems; The MYCIN Experiments of the Stanford Heuristic Programming Project, Reading, Massuchusetts, Addison Wesley.

Demski, T.S. and Kreps, D.M. (1980), "Models in Managerial Accounting", Journal of Accounting Research, 20 Supplement, 117-148.

Dittman and Prakash (1978), "Cost Variance Investigation Markovian Control of Markov Process", Journal of Accounting Research, 16, 14-25.

Frick, H. (1977), "Game Theoretic Treatment of Material Accountability Problems: Part II", International Journal of Game Theory, 6, 41-49.

Gettys, C.F. and Wilke, T.A. (1969), "Application of Bayes Theorem When the True Data State is Uncertain", Organization Behaviour and Human Performance, 4, 125-141.

Horngren, C.T. (1977), Cost Accounting: A Managerial Emphasis, 4th ed. Prentice Hall.

Kaplan, R.S. (1975), "The Significance and Investigation of Cost Variance: Survey and Extensions", Journal of Accounting Research, 13, 311-337.

Shortliffe, E.H. and Buchanan, B.G. (1984), "A Model of Inexact Reasoning in Medicine" in Buchanan and Shortliffe(1984).

Modo Company*

August Germain, a director of Modo Company, received the following Exhibits 1 through 4 as part of the packet of material sent in advance of the April board meeting. As 1 of 15 directors, he thought that in [illegible]

EXHIBIT 1 Operations Highlights: Net Sales Summary, March ($000)

Current month				*Budget period*		
Budget	*Actual*	*Increase/ (decrease) over budget (percent)*	*Division*	*Budget*	*Actual*	*Increase/ (decrease) over budget (percent)*
$ 2,550	$ 3,816	50%	A	$ 8,345	$10,483	26%
1,155	24	(98)	B	3,398	65	(98)
0	59	0	C	0	276	0
3,274	3,636	11	D	9,443	8,598	(9)
1,781	1,468	(17)	E	4,149	3,874	(7)
2,843	2,692	(5)	F	8,825	9,458	7
6,515	6,101	(6)	G	19,041	16,664	(12)
800	1,067	33	H	2,000	2,912	46
129	0	(100)	J	333	0	(100)
0	0	0	K	0	0	0
591	201	(66)	L	1,966	1,568	(20)
455	667	27	M	1,075	1,363	27
491	615	25	N	1,712	1,357	(21)
4,885	5,574	14	O	12,935	14,277	10
3,514	2,557	(27)	P	11,677	7,966	(32)
4,241	2,200	(48)	Q	10,036	7,354	(27)
492	557	13	R	1,411	1,623	15
33,686	31,134	(8)	Total net sales	96,346	87,858	(9)
(20)	(73)	(265)	Less: Interdivisional sales	(70)	(326)	366
$ 33,666	$ 31,061	(8)%	Consolidated net sales	$96,276	$87,532	(9)%

Note: Unfavorable variances are in parentheses.
All dollar amounts in thousands.

EXHIBIT 2 Operations Highlights: Division Pretax Income Summary, March ($000)

Current month				*Budget period*		
Budget	*Actual*	*Variance*	*Division*	*Budget*	*Actual*	*Variance*
$ (4)	$ 213	$ 217	A	$ (10)	$ 502	$ 512
(248)	(103)	145	B	(937)	(418)	519
0	(21)	(21)	C	0	(39)	(39)
362	357	(5)	D	901	550	(351)
471	351	(120)	E	980	937	(43)
377	452	75	F	1,244	1,510	266
772	881	109	G	1,698	1,620	(78)
82	103	21	H	130	185	55
(13)	0	13	J	(41)	0	41
0	0	0	K	0	0	0
(137)	(210)	(73)	L	(265)	(266)	(1)
57	28	(29)	M	88	0	38
(84)	(94)	(10)	N	(130)	(235)	(105)
624	636	12	O	1,440	2,419	979
(122)	(337)	(215)	P	(117)	(776)	(659)
62	(612)	(674)	Q	(493)	(1,467)	(974)
0	38	38	R	0	206	206
8	4	(4)	S	26	65	39
$2,207	$1,686	$(521)	Division pretax income (loss)	$4,714	$5,117	$ 403

Note: Unfavorable variances are in parentheses.

*This case was prepared by R. N. Anthony, Harvard Business School.
Copyright © by the President and Fellows of Harvard College
Harvard Business School case 180-060.

EXHIBIT 3 Operations Highlights: Trade Accounts Receivable, March ($000)

					Number of days sales* in receivables	
Last year	Budget	Actual	Variance	Division	Last year days	Actual days
$ 804	$ 722	$ 4,309	$(3,587)	A	6	41
3,353	3,051	2,831	420	B	65	58
122	—	0	—	C	25	—
3,801	610	4,011	(3,401)	D	37	38
2,590	3,493	3,457	36	E	122	177
13,051	13,810	12,654	1,156	F	161	154
1,632	1,675	1,677	(2)	H and J	61	55
2,113	2,500	2,393	107	K	40	41
8,023	8,474	8,972	(498)	O	59	63
9,077	9,065	8,282	783	P	69	61
8,316	9,213	6,781	2,432	Q	56	46
52,682	52,613	55,167	(2,554)	Total		
12,857	14,634	12,457	2,177	G		
65,539	67,247	67,624	$ (377)	Total		
2,950	—	103		Discontinued divisions	73	77
$68,489	$67,247	$67,[illegible]27		Total	64	65

*Using net sales for the most recent 12-month period.

EXHIBIT 4 Operations Highlights: Inventories Summaries, March ($000)

Inventories, gross					Net sales* to ending inventory	
Last year	Budget	Actual	Variance	Division	Last year times	Actual times
$ 8,857	$ 9,050	$ 7,626	$ 1,424	A	4.1	5.1
5,108	3,412	3,423	(11)	B	3.7	4.8
721	—	0	—	C	2.4	—
9,608	9,702	8,133	1,569	D	3.9	4.8
1,440	1,612	1,427	185	E	5.4	5.0
6,905	7,091	6,480	611	F	4.3	4.6
903	750	889	(139)	H	10.8	12.5
3,031	2,350	3,433	(1,083)	L	6.3	6.2
9,593	12,280	12,241	39	O	5.2	4.3
14,173	14,010	15,419	(1,409)	P	3.4	3.2
11,972	8,540	9,250	(710)	Q	4.5	5.8
72,3[illegible]	68,797	68,321	476	Total		
14,288	14,016	16,013	(1,997)	International		
86,597	82,813	84,334	$(1,521)	Total		
3,908	—	—		Discontinued divisions	4.5	3.7
$90,505	$82,813	$84,334		Total	4.3	4.5

*Using net sales for the most recent 12-month period.

fairness to the others, he should ask not more than three questions about these data.

Question

What questions should he ask?

APPENDIX TWO

SAMPLE RULEBASE FOR MODO COMPANY CASE

```
/* rules for investigating a division */

rule(1,"
    probe(Div) if
        probe(income,Div).
    ",100).  /* Probe division if income variance significant*/

rule(2,"
    probe(Div) if
        probe(sales,Div).
    ",100).  /* Probe division if sales variance significant*/

rule(3,"
    probe(Div) if
        probe(ar,Div).
    ",50).  /* Probe division if A/R variance significant*/

rule(4,"
    probe(Div) if
        probe(invy,Div).
    ",50). /* Probe division if inventory variance significant*/

/*rules for investigating income variances */

rule(10,"
    probe(income,Div) if
        variance(income,Div,Mvar,Qvar,Mpct,Qpct),
        Mvar < 0,
        fuzzy(ramp(-Mvar,25,100) or ramp(-Mpct,5,20)).
    ",100). /* large negative monthly variance*/

rule(20,"
    probe(income,Div) if
        variance(income,Div,Mvar,Qvar,Mpct,Qpct),
        Qvar < 0, Mvar < 0,
        Moderate is ramp(-Mvar,10,50) or ramp(-Mpct,3,10),
        Large is ramp(-Qvar,50,200) or ramp(-Qpct,5,20),
        fuzzy(Large and Moderate).
    ",50). /* large quarterly with moderate monthly variance*/

rule(30,"
    probe(income,Div) if
        variance(income,Div,Delta_profit,_,_,_),
        variance(sales,Div,Delta_sales,_,_,_),
        Ratio is pct(Delta_profit,Delta_sales),
        margin_squeeze(Ratio,Delta_sales).
    ",100).  /* margin squeeze in monthly data */

rule(40,"
    probe(income,Div) if
```

```
          variance(income,Div,_,Delta_profit,_,_),
          variance(sales,Div,_,Delta_sales,_,_),
          Ratio is pct(Delta_profit,Delta_sales),
          margin_squeeze(Ratio,Delta_sales).
      ",50). /* margin squeeze in quarterly data */


/* margin squeeze used in income variance rules */

rule(110,"
     margin_squeeze(Ratio,Delta_sales) if
          Delta_sales > 0,
          fuzzy(ramp(Ratio,5,-30)).
     ",100). /* incremental sales have not earned
                even 5% contribution */

rule(115,"
     margin_squeeze(Ratio,Delta_sales) if
          Delta_sales < 0,
          fuzzy(ramp(Ratio,25,80)).
     ",100). /* contribution lost exceeds 25% of sales drop */


/* rules for sales variances */

rule(510,"
     probe(sales,Div) if
          variance(sales,Div,Mvar,Qvar,Mpct,Qpct),
          Mvar < 0,
          fuzzy(ramp(-Mvar,100,600) or ramp(-Mpct,5,20)).
     ",100).  /* large negative monthly variance */

rule(520,"
     probe(sales,Div) if
          variance(sales,Div,Mvar,Qvar,Mpct,Qpct),
          Mvar < 0, Qvar < 0,
          Moderate is ramp(-Mvar,50,300) or ramp(-Mpct,3,10),
          Large is ramp(-Qvar,200,1200) or ramp(-Qpct,5,20),
          fuzzy(Moderate and Large).
     ",50).  /* large quarterly with moderate monthly variance */

rule(530,"
     probe(sales,Div) if
          variance(sales,Div,Mvar,Mpct,_,_),
          Mvar < 0,
          wc_variance(ar,Div,_,Pct),
          fuzzy(ramp(Pct,5,20) and
               (ramp(-Mvar,50,300) or ramp(-Mpct,3,10))).
     ",50).  /* negative sales variance with lengthening ACP*/


/*rules for receivable variances */

rule(610,"
     probe(ar,Div) if
```

```
          wc_variance(ar,Div,Amt,Pct),
          fuzzy(ramp(Amt,150,1000) or ramp(Pct,5,20)).
     ",100).  /* large variance */


/*rules for inventory variances */

rule(710,"
     probe(invy,Div) if
          wc_variance(invy,Div,Amt,Pct),
          fuzzy(ramp(Amt,150,1000) or ramp(Pct,5,20)).
     ",100).  /* large variance */

rule(1000,"
     probe(_,c).
     ",-80).  /* ignore Division c variances (being closed) */

question(100,"
     variance(Item,Div,Mvar,Qvar,Mpct,Qpct) if
          bound(Item), bound(Div),
          write('Data for '), write(Div),
          write(' not available'),nl,
          write('Enter Mvar :'),read(Mvar),nl,
          write('Enter Mpct :'),read(Mpct),nl,
          write('Enter Qvar :'),read(Qvar),nl,
          write('Enter Qpct :'),read(Qpct),nl.
     ",100). /* ask user in case data does not exist */
```

APPENDIX THREE

LOG OF SAMPLE SESSION

(Using the Rulebase Listed in Appendix Two)

```
Goal: probe(income,_)
      /*Which income variances should we probe?*/
probe(income,e).  Certainty Factor 100%
Rule #10  CFs : Rule=100% Prem=100% Effv=100% Cum=100%
Rule #30  CFs : Rule=100% Prem=29% Effv=29% Cum=100%

probe(income,l).  Certainty Factor 64%
Rule #10  CFs : Rule=100% Prem=64% Effv=64% Cum=64%

probe(income,m).  Certainty Factor 100%
Rule #10  CFs : Rule=100% Prem=100% Effv=100% Cum=100%
Rule #20  CFs : Rule=50% Prem=100% Effv=50% Cum=100%
Rule #30  CFs : Rule=100% Prem=89% Effv=89% Cum=100%
Rule #40  CFs : Rule=50% Prem=100% Effv=50% Cum=100%

probe(income,p).  Certainty Factor 100%
Rule #10  CFs : Rule=100% Prem=100% Effv=100% Cum=100%
Rule #20  CFs : Rule=50% Prem=100% Effv=50% Cum=100%

probe(income,q).  Certainty Factor 100%
Rule #10  CFs : Rule=100% Prem=100% Effv=100% Cum=100%
Rule #20  CFs : Rule=50% Prem=100% Effv=50% Cum=100%
Rule #30  CFs : Rule=100% Prem=15% Effv=15% Cum=100%
Rule #40  CFs : Rule=50% Prem=20% Effv=10% Cum=100%

probe(income,d).  Certainty Factor 30%
Rule #30  CFs : Rule=100% Prem=17% Effv=17% Cum=17%
Rule #40  CFs : Rule=50% Prem=31% Effv=16% Cum=30%

probe(income,n).  Certainty Factor 40%
Rule #30  CFs : Rule=100% Prem=37% Effv=37% Cum=37%
Rule #40  CFs : Rule=50% Prem=9% Effv=5% Cum=40%

7 Solutions
There were also 2 solutions with Cf ≤ cutoff of 20
```

```
Goal: <ESC>

Command: view /*Sort according to certainty factors*/
probe(income,e).  Certainty Factor 100%
Rule #10  CFs : Rule=100% Prem=100% Effv=100% Cum=100%
Rule #30  CFs : Rule=100% Prem=29% Effv=29% Cum=100%

probe(income,m).  Certainty Factor 100%
Rule #10  CFs : Rule=100% Prem=100% Effv=100% Cum=100%
Rule #20  CFs : Rule=50% Prem=100% Effv=50% Cum=100%
Rule #30  CFs : Rule=100% Prem=89% Effv=89% Cum=100%
Rule #40  CFs : Rule=50% Prem=100% Effv=50% Cum=100%

probe(income,p).  Certainty Factor 100%
Rule #10  CFs : Rule=100% Prem=100% Effv=100% Cum=100%
Rule #20  CFs : Rule=50% Prem=100% Effv=50% Cum=100%

probe(income,q).  Certainty Factor 100%
Rule #10  CFs : Rule=100% Prem=100% Effv=100% Cum=100%
Rule #20  CFs : Rule=50% Prem=100% Effv=50% Cum=100%
Rule #30  CFs : Rule=100% Prem=15% Effv=15% Cum=100%
Rule #40  CFs : Rule=50% Prem=20% Effv=10% Cum=100%

probe(income,l).  Certainty Factor 64%
Rule #10  CFs : Rule=100% Prem=64% Effv=64% Cum=64%

probe(income,n).  Certainty Factor 40%
Rule #30  CFs : Rule=100% Prem=37% Effv=37% Cum=37%
Rule #40  CFs : Rule=50% Prem=9% Effv=5% Cum=40%

probe(income,d).  Certainty Factor 30%
Rule #30  CFs : Rule=100% Prem=17% Effv=17% Cum=17%
Rule #40  CFs : Rule=50% Prem=31% Effv=16% Cum=30%

probe(income,c).  Certainty Factor 20%
Rule #10  CFs : Rule=100% Prem=100% Effv=100% Cum=100%
Rule #20  CFs : Rule=50% Prem=100% Effv=50% Cum=100%
Rule #30  CFs : Rule=100% Prem=100% Effv=100% Cum=100%
Rule #40  CFs : Rule=50% Prem=54% Effv=27% Cum=100%
NEGATIVE Rule #1000  CFs : Rule=-80% Prem=100% Effv=-80% Cum=20%

probe(income,o).  Certainty Factor 9%
Rule #30  CFs : Rule=100% Prem=9% Effv=9% Cum=9%
```

```
Goal: <ESC>

Command: trace_all
     /* We ask for a full trace before issuing next command */
Goal: <ESC>

Command: justify(probe(income,d))
/* Since D's variance is so small we are surprised by the
     high certainty factor and ask for a full justification */
**** Justification of probe(income,d) *****
Rule #30  CFs : Rule=100% Prem=17% Effv=17% Cum=17%
Premises :
   variance(income,d,-5,-351,-1,-39)
   variance(sales,d,362,-845,11,-9)
   -1 is pct(-5,362)
      pct(-5,362) = -1
   margin_squeeze(-1,362)
      Rule #110  CFs : Rule=100% Prem=17% Effv=17% Cum=17%
      Premises :
         362 > 0
         fuzzy(ramp(-1,5,-30))
            ramp(-1,5,-30) = 17
Rule #40  CFs : Rule=50% Prem=31% Effv=16% Cum=30%
Premises :
   variance(income,d,-5,-351,-1,-39)
   variance(sales,d,362,-845,11,-9)
   42 is pct(-351,-845)
      pct(-351,-845) = 42
   margin_squeeze(42,-845)
      Rule #115  CFs : Rule=100% Prem=31% Effv=31% Cum=31%
      Premises :
         -845 < 0
         fuzzy(ramp(42,25,80))
            ramp(42,25,80) = 31
```

```
Goal: <ESC>

Command: refute(probe(income,1))
/* We expect a higher certainty factor for L's income
        variance and want to know why the other rules failed */
**** Failures of probe(income,1) *****
probe(income,1)  succeeded with Cf = 64%
BUT FOLLOWING RULES FAILED
    Rule #20 failed.  Failing Clause :
    fuzzy(Large and Moderate)
    i.e.  fuzzy(0 and 100)
        Failure of fuzzy(0 and 100)
        Reason  : SUCCEEDED WITH ZERO CF
            0 and 100 = 0
    Rule #30 failed.  Failing Clause :
    margin_squeeze(Ratio,Delta_sales)
    i.e.  margin_squeeze(19,-390)
        Failure of margin_squeeze(19,-390)
        Reason  : ALL RULES FAILED
            Rule #110 failed.  Failing Clause :
            Delta_sales > 0
            i.e.  -390 > 0
                Failure of -390 > 0
                Reason  : PREDICATE FAILED
            Rule #115 failed.  Failing Clause :
            fuzzy(ramp(Ratio,25,80))
            i.e.  fuzzy(ramp(19,25,80))
                Failure of fuzzy(ramp(19,25,80))
                Reason  : SUCCEEDED WITH ZERO CF
                    ramp(19,25,80) = 0
    Rule #40 failed.  Failing Clause :
    margin_squeeze(Ratio,Delta_sales)
    i.e.  margin_squeeze(0,-398)
        Failure of margin_squeeze(0,-398)
        Reason  : ALL RULES FAILED
            Rule #110 failed.  Failing Clause :
            Delta_sales > 0
            i.e.  -398 > 0
                Failure of -398 > 0
                Reason  : PREDICATE FAILED
            Rule #115 failed.  Failing Clause :
            fuzzy(ramp(Ratio,25,80))
            i.e.  fuzzy(ramp(0,25,80))
                Failure of fuzzy(ramp(0,25,80))
                Reason  : SUCCEEDED WITH ZERO CF
                    ramp(0,25,80) = 0
```

```
Goal: probe(o)
     /* Should division O be probed ? */
probe(o).  Certainty Factor 65%
Rule #1  CFs : Rule=100% Prem=9% Effv=9% Cum=9%
Premises :
   probe(income,o)
      Rule #30  CFs : Rule=100% Prem=9% Effv=9% Cum=9%
      Premises :
         variance(income,o,12,979,2,68)
         variance(sales,o,689,1342,14,10)
         2 is pct(12,689)
            pct(12,689) = 2
         margin_squeeze(2,689)
            Rule #110  CFs : Rule=100% Prem=9% Effv=9% Cum=9%
            Premises :
               689 > 0,
               fuzzy(ramp(2,5,-30))
                  ramp(2,5,-30) = 9
Rule #3  CFs : Rule=50% Prem=46% Effv=23% Cum=30%
Premises :
   probe(ar,o)
      Rule #610  CFs : Rule=100% Prem=46% Effv=46% Cum=46%
      Premises :
         wc_variance(ar,o,538,7)
         fuzzy(ramp(538,150,1000) or ramp(7,5,20))
            ramp(538,150,1000) or ramp(7,5,20) = 46
               ramp(538,150,1000) = 46
               ramp(7,5,20) = 13
Rule #4  CFs : Rule=50% Prem=100% Effv=50% Cum=65%
Premises :
   probe(invy,o)
      Rule #710  CFs : Rule=100% Prem=100% Effv=100% Cum=100%
      Premises :
         wc_variance(invy,o,2203,21)
         fuzzy(ramp(2203,150,1000) or ramp(21,5,20))
            ramp(2203,150,1000) or ramp(21,5,20) = 100
               ramp(2203,150,1000) = 100
               ramp(21,5,20) = 100

True.
```

```
Goal: <ESC>

Command: trace(1) /* Ask for brief proof traces only */
Goal: probe(sales,_)
      /*Which sales variances should we probe?*/
probe(sales,b).  Certainty Factor 100%
Rule #510  CFs : Rule=100% Prem=100% Effv=100% Cum=100%
Rule #520  CFs : Rule=50% Prem=100% Effv=50% Cum=100%

probe(sales,e).  Certainty Factor 91%
Rule #510  CFs : Rule=100% Prem=80% Effv=80% Cum=80%
Rule #520  CFs : Rule=50% Prem=13% Effv=7% Cum=81%
Rule #530  CFs : Rule=50% Prem=100% Effv=50% Cum=91%

probe(sales,g).  Certainty Factor 82%
Rule #510  CFs : Rule=100% Prem=63% Effv=63% Cum=63%
Rule #520  CFs : Rule=50% Prem=100% Effv=50% Cum=82%

probe(sales,l).  Certainty Factor 100%
Rule #510  CFs : Rule=100% Prem=100% Effv=100% Cum=100%
Rule #520  CFs : Rule=50% Prem=100% Effv=50% Cum=100%

probe(sales,p).  Certainty Factor 100%
Rule #510  CFs : Rule=100% Prem=100% Effv=100% Cum=100%
Rule #520  CFs : Rule=50% Prem=100% Effv=50% Cum=100%

probe(sales,q).  Certainty Factor 100%
Rule #510  CFs : Rule=100% Prem=100% Effv=100% Cum=100%
Rule #520  CFs : Rule=50% Prem=100% Effv=50% Cum=100%

6 Solutions
There was also 1 solution with Cf ≤ cutoff of 20
Goal: <ESC>
```

```
Command: view   /*Sort according to certainty factors*/
probe(sales,b).  Certainty Factor 100%
Rule #510  CFs : Rule=100% Prem=100% Effv=100% Cum=100%
Rule #520  CFs : Rule=50% Prem=100% Effv=50% Cum=100%

probe(sales,l).  Certainty Factor 100%
Rule #510  CFs : Rule=100% Prem=100% Effv=100% Cum=100%
Rule #520  CFs : Rule=50% Prem=100% Effv=50% Cum=100%

probe(sales,p).  Certainty Factor 100%
Rule #510  CFs : Rule=100% Prem=100% Effv=100% Cum=100%
Rule #520  CFs : Rule=50% Prem=100% Effv=50% Cum=100%

probe(sales,q).  Certainty Factor 100%
Rule #510  CFs : Rule=100% Prem=100% Effv=100% Cum=100%
Rule #520  CFs : Rule=50% Prem=100% Effv=50% Cum=100%

probe(sales,e).  Certainty Factor 91%
Rule #510  CFs : Rule=100% Prem=80% Effv=80% Cum=80%
Rule #520  CFs : Rule=50% Prem=13% Effv=7% Cum=81%
Rule #530  CFs : Rule=50% Prem=100% Effv=50% Cum=91%

probe(sales,g).  Certainty Factor 82%
Rule #510  CFs : Rule=100% Prem=63% Effv=63% Cum=63%
Rule #520  CFs : Rule=50% Prem=100% Effv=50% Cum=82%

probe(sales,f).  Certainty Factor 10%
Rule #510  CFs : Rule=100% Prem=10% Effv=10% Cum=10%
```

```
Goal: probe(_)  /*Which divisions should we probe?*/

   ~ ~ ~ ~ ~ ~ ~ ~ system response deleted ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
   ~ ~ ~ ~ ~ ~ ~ ~                         ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
Goal: <ESC>

Command: view  /*Sort according to certainty factors*/
probe(e).  Certainty Factor 100%
Rule #1  CFs : Rule=100% Prem=100% Effv=100% Cum=100%
Rule #2  CFs : Rule=100% Prem=91% Effv=91% Cum=100%
Rule #3  CFs : Rule=50% Prem=100% Effv=50% Cum=100%
Rule #4  CFs : Rule=50% Prem=13% Effv=7% Cum=100%

probe(l).  Certainty Factor 100%
Rule #1  CFs : Rule=100% Prem=64% Effv=64% Cum=64%
Rule #2  CFs : Rule=100% Prem=100% Effv=100% Cum=100%

probe(m).  Certainty Factor 100%
Rule #1  CFs : Rule=100% Prem=100% Effv=100% Cum=100%

probe(p).  Certainty Factor 100%
Rule #1  CFs : Rule=100% Prem=100% Effv=100% Cum=100%
Rule #2  CFs : Rule=100% Prem=100% Effv=100% Cum=100%
Rule #4  CFs : Rule=50% Prem=91% Effv=46% Cum=100%

probe(q).  Certainty Factor 100%
Rule #1  CFs : Rule=100% Prem=100% Effv=100% Cum=100%
Rule #2  CFs : Rule=100% Prem=100% Effv=100% Cum=100%

probe(b).  Certainty Factor 100%
Rule #2  CFs : Rule=100% Prem=100% Effv=100% Cum=100%

probe(g).  Certainty Factor 82%
Rule #2  CFs : Rule=100% Prem=82% Effv=82% Cum=82%

probe(o).  Certainty Factor 65%
Rule #1  CFs : Rule=100% Prem=9% Effv=9% Cum=9%
Rule #3  CFs : Rule=50% Prem=46% Effv=23% Cum=30%
Rule #4  CFs : Rule=50% Prem=100% Effv=50% Cum=65%

probe(a).  Certainty Factor 50%
Rule #3  CFs : Rule=50% Prem=100% Effv=50% Cum=50%

probe(n).  Certainty Factor 40%
Rule #1  CFs : Rule=100% Prem=40% Effv=40% Cum=40%

probe(d).  Certainty Factor 30%
Rule #1  CFs : Rule=100% Prem=30% Effv=30% Cum=30%

probe(c).  Certainty Factor 20%
Rule #1  CFs : Rule=100% Prem=20% Effv=20% Cum=20%

probe(f).  Certainty Factor 10%
Rule #2  CFs : Rule=100% Prem=10% Effv=10% Cum=10%
```