# SOME EXPERIENCES WITH EXPERT-2

C.M. Sargent, L.G. Watson, R.M. Westman
Department of Mechanical Engineering
University of Saskatchewan
Saskatoon, Saskatchewan, Canada S7N 0W0

**ABSTRACT**

The use of FORTH to implement knowledge-based (expert) systems is seldom endorsed by the Artificial Intelligence community. However, FORTH's speed, extensibility, portability, and rapid-prototyping environment make it ideally suited for this task, especially on microcomputer-based systems. One example of using FORTH as a basis for expert system development is a very useful tool kit called EXPERT-2 (Park, Mountain View Press).

Although EXPERT-2 is intended to be a learning aid only, it is much more, because the full FORTH language is accessible through the RUN-type operators. Also, since the source code is included, programmers can tailor the tool kit to a specific application, or add features that are not presently available. Thus, EXPERT-2 provides the basis for an extremely powerful, flexible expert system tool kit in FORTH.

Our experience with EXPERT-2 is described on two levels: 1) modifications to the tool kit source code itself, and 2) exploitation of the syntactical and structural constraints within the program environment. Among the useful modifications are the additions of tracing and debugging facilities, a simple forward reasoning scheme, additional operators, and the ability to chain several rule-bases together in one consultation.

## INTRODUCTION

This paper presupposes familiarity with, or at least access to EXPERT-2. EXPERT-2 is a fifth-generation computer language which facilitates the creation of knowledge-based computer programs. It was meant to be an introduction to expert systems in FORTH. However, because EXPERT-2 is written in FORTH and allows access to the underlying FORTH system, it provides a stepping stone to more elaborate and sophisticated expert systems in FORTH.

Since the source code is included in the documentation accompanying EXPERT-2, virtually any desired feature can be implemented. We have made various additions to EXPERT-2 over the past 2 years that have proved to be extremely useful, some of which are described below.

These additional features are presented to demonstrate some of the tools which make development of FORTH-based expert systems practical. Other features that might also be useful but are not yet implemented are: approximate reasoning methods, data-base interaction, and real-time data acquisition and control facilities.

## FORWARD REASONING

As a consequent-reasoning system, EXPERT-2 chooses a goal (hypothesis) and works backwards through its rules trying to prove it. If a proof cannot be found, execution moves to the next hypothesis. In

contrast, an antecedent (forward) reasoning system starts with facts and tries to establish a conclusion. Ideally, both methods would be employed in one inference engine.

A simple forward-reasoning scheme was developed through the addition of **SELECT** and **THENN**. **SELECT** offers the opportunity to begin with some known facts. When **SELECT** is executed, all **THENN** strings are presented with the question:

IS THIS TRUE? (Y=YES, N=NO, RETURN=NOT KNOWN)

Returning 'YES' or 'NO' stores the string on the **KNOWNTRUE** or **KNOWNFALSE** stack, respectively. After all **THENN** strings have been presented, execution passes to **DIAGNOSE+** (**DIAGNOSE** without **UNLEARN**) for normal EXPERT-2 testing of hypotheses. **THENN** is merely used to distinguish strings for use with **SELECT**. During **DIAGNOSE**, **THENN** behaves just like **THEN**.

This method allows the user to volunteer known information at the beginning of a session. As a result, irrelevant questions are minimized and hypothesis testing can proceed more efficiently.

**TRACE/DEBUGGING FACILITIES**

A TRACE facility was added to provide the user with the ability to follow the reasoning leading to the current question or conclusion in a consultation. The format for questions was modified to include the trace option as shown here:

IS THIS TRUE? (Y=YES, N=NO, W=WHY, T=TRACE)

Selecting 'T' for trace presents each rule in the chain of reasoning with IS TRUE, IS FALSE, or IS NOT KNOWN following each statement, based on the user's responses and the program's deductions. After each traced rule is presented, the user is asked if he wants to continue the trace. Responding 'YES' presents the next rule in the current reasoning chain. A 'NO' response to the trace query will switch execution back to the current question immediately. When all rules have been presented, the original question is repeated. This trace feature is also present at the end of a session, once a hypothesis has been verified.

Several of the additions to EXPERT-2 were debugging tools. Extensive use was made of FORTH words that listed important parameters, for instance: **LISTHYP** lists the rejected hypotheses, **DISPLAY** lists the **KNOWNTRUE** and **KNOWNFALSE** stacks, and **LISTIF** and **LISTTHEN** will detect if a rule has more than one **IF** or **THEN** statement. This last situation is undesirable since the inference engine will see the second **IF** as a **THEN**, creating confusion in further testing.

Another debugging tool that found extensive use was **MORERULES**. It allows the compilation of additional rules without forgetting **WALL** and starting over. It switches in the **RULES** vocabulary and removes the last set of zeros from **RULESTK**. A **DONE** must complete the additional rules to return the context vocabulary to **FORTH**. In order to add rules to the presently loaded rule-base, the screens with additional rules are **LOAD**ed, preceeded by **MORERULES** and succeeded by **DONE**. For additions to lengthy rule-bases, this feature is a real time-saver.

**ADDITIONAL OPERATORS**

In the knowledge encoding process, there may be more than one method of arriving at the same conclusion. This situation results when two or more rules have different antecedents but the same consequent.

While the order in which rules are listed does not affect the execution of the program (since the OR between rules with identical consequents is implied in the program), the logic may be difficult to follow if rules which have the same consequent are not grouped together in the rule-base.

For this reason the **OR** operator was added. It is used to gather all the antecedent combinations together into one rule. The logic of the single rule approach is easier to follow when 'WHY' is invoked during the user session, because all possible antecedents are presented in one rule.

For similar reasons, **ORIF** and **ORIFNOT** operators were also added. The operation is parallel to that of the **OR**, but **ORIF** and **ORIFNOT** only work between two antecedent strings, whereas **OR** works between two sets of antecedents.

Another operator addition was **ANDTHENNOT**. It is used to provide a negative consequent, and is used only after a **THEN**. If two consequents are mutually exclusive, then choosing one eliminates the other one. Thus, if a rule which includes an **ANDTHENNOT** is triggered, the string preceeded by the **ANDTHENNOT** is put on the **KNOWNFALSE** stack. This eliminates from further testing any rule or hypothesis which uses that string as a positive context antecedent.

**MULTIPLE RULE-BASES**

The operator **ADDRULES** was created to facilitate the compilation of rules only as they are required during a consultation. For example, in the classic ANIMALS game, some initial rules would decide if the animal was mammal, reptile, or bird. If it was deduced that the animal was a bird, the BIRD rules would then be compiled into memory from disk. Thus, these rules would occupy memory only if they are needed. They can be called from a rule with a **RUN** operator, such as **ANDTHENRUN ADDRULES**, where **ADDRULES** would load the appropriate screens from disk.

The purpose for this methodology is twofold. First, it is useful for small machines where the available memory restricts the number of rules that may be compiled at one time. Secondly, it provides a modular approach for creating large rule-bases.

**EXPLOITATION OF THE EXPERT-2 ENVIRONMENT**

To minimize the search time involved with large rule-bases, a combination of menus and variables was implemented to flag only certain hypotheses for testing. This technique provides a coarse filter for establishing initially valid hypotheses. Consequently, the number of rules that must be considered is reduced.

Before the rules are compiled, a variable is defined and initialized for each hypothesis in the rule-base. The user is presented with a menu or set of menus that classify the hypotheses into broad groups. Based on the user's selection from this menu system, each variable is updated to represent the suitability (either true or false) of its corresponding hypothesis for that selection.

To use the contents of this variable in the rule-base, each rule containing a hypothesis would also contain an **IFRUN** antecedent. This operator would execute a FORTH word to fetch the value from its corresponding variable. If the value is a true flag, testing proceeds to the next antecedent in the hypothesis rule. If the value is a false flag, that hypothesis is rejected, and testing proceeds to the next

hypothesis in the rule-base.

Another significant development in the EXPERT-2 environment was the use of default structures. In the decision tree of rules, once selection has proceeded down one branch, it is often desirable to force a selection from that branch. In the ANIMALS program, if it has already been deduced that the animal is a bird and a suitable bird cannot be found, the inference engine should be restrained from trying to prove that the animal is a cow. This situation arises since testing of hypotheses proceeds sequentially as they occur on the **HYPSTACK.**

Several methods were tried to effect a default in certain branches of the rule-base. Essentially, the structure was to provide a way of choosing the first hypothesis in the branch after all other hypotheses in the branch were exhausted. This would force a conclusion from that branch even if all hypotheses in the branch were exhausted.

One solution was to add another hypothesis to the **HYPSTACK** immediately following the hypotheses for a particular branch. This hypothesis would have the same character string as the first hypothesis in the branch, however the character string of the last hypothesis would be separated from the **THENHYP** operator by two spaces instead of one. A string with two spaces between two words is regarded as entirely different from the same string with only one space between the same two words. This exploitation of the syntax requirements in EXPERT-2 makes it possible to have two hypotheses that look virtually the same to the user, but each serves a different purpose. The first hypothesis is used in the usual way, and the second hypothesis is used as the default for the end of the branch.

If desired, the user can be informed that the verifed hypothesis is a default by using an **ANDTHENRUN** operator to invoke an explanation. This makes it easier for the user to follow the logic of the decision in the program.

Another default structure was to control testing in the branch by a rule that would verify the first hypothesis if its antecedents were true or if the hypotheses for the other choices in the branch could not be proven. For example, if hypotheses A, B, and C composed one branch of rules and A was to be the default, a rule for A would only be satisfied if the antecedents for A were true or if B and C could not be proven. By placing this rule higher in the rule-base than the rules for B and C, testing of B and C would result from a rule for A and not from the position of B and C on the **HYPSTACK.** If B or C is proven as a result of the rule for A, then there is no need for the default.

## CONCLUSIONS

EXPERT-2 has been in use in the Department of Mechanical Engineering for the past two years. Some of the research projects that are under development include the selection of wrought stainless steels, a mechanical shaft design program, self-diagnosis of a hydraulic pump, hydraulic circuit design, stress analysis, and a diving coach expert system.

It has been our experience that some very difficult problems can be solved with an expert system on a small computer. FORTH is ideally suited for this task, however proper tools are needed. EXPERT-2 provides a basis for creating these tools. To date, the limitations imposed by using a binary reasoning mechanism remain as the biggest drawback to using EXPERT-2 for problem domains with inexact solutions.