Expert Systems in Forth

Jack Park

This issue provides a diverse (though not complete) discussion of approaches to implementation of expert systems in the Forth environment. A look at two medical applications of these implementations is also provided. A discussion of parsing and a simple implementation are included. Finally a technical note details a fuzzy reasoning reference engine.

Fundamentally, we tend to think that humans solve problems by combinations of data (event or state), and by a goal directed search for solutions. For completeness, however, it should be noted that the debate on the issue of how humans think includes issues of analogical or metaphorical thinking. When implementing these approaches to problem solving, we attach the names "forward chaining" to data driven expert systems, and "backward chaining" to the goal driven systems. The specific part of the expert system code which performs the backward or forward search is usually called the "inference engine" and the portion of the code written by either the "expert" or a programmer trained in capturing expertise in a program (a knowledge engineer) is called a knowledge base. Perhaps more than any other distinction possible, it is the separation of the knowledge base from the inference engine that evolved the style of programming we call Expert Systems.

Implementations

The implementation issues focus not on whether Forth is a suitable environment for the work, but on how certain aspects of the user interface—specifically, the interface with the knowledge engineer, the rule writer—is structured. Fundamentally, we find two arguments: the argument that the rule structure should be free, English-like, and understandable by the novice user; and the argument that the rule structure should adhere as closely to the Forth structure—and syntax—as possible. With tongue in cheek, I admit that I started with the argument that the English-like, easily readable approach was appropriate for personal computer-based expert systems, and this was the original approach taken in EXPERT-2 [PAR84]. A later system (EXPERT-4) carried this approach further and greatly simplified the user interface. However, recalling that tongue is firmly ensconced in cheek, I have migrated toward a much more complicated user interface with EXPERT-5, one much like a post-fix OPS5 and not unlike the FORPS system discussed in this issue. But I must qualify this migration since it was made in an effort to gain computational speed so that a conversational natural-language front end could be implemented in the same environment that the expert system knowledge base would operate.

Gains in computational speed come from reduction in computational activity necessary to accomplish any task. Thus, Dana Redington has argued (in this issue) that the Forth `NEXT` is an appropriate inference engine. Indeed, the evolution of Forth engines that bury `NEXT` as a free part of the execution cycle suggests that the closer one approaches the Forth environment, the faster one might perform logical computations necessary for inferencing. Computational speed, of course, is not the entire issue, but one worth considering. Thus, the arguments for rule-writing style must be weighted by external issues.

Like the Basic compiler [PER82] that compiles Basic down to Forth, compilers can be written to turn conversational-style knowledge bases into Forth programs [FEY84]. Another approach is run-time parsing of a natural language conversation (this issue). Indeed, there are a number of approaches left unexplored in this issue of JFAR.

Looking at the rule structures of a few inference engines, we can see quite graphically how various designers approach the user interface. In the original EXPERT-2, a typical animals game rule looked like:

```
IF animal has feathers
ANDRUN weight>2kg
THEN animal is a bird
```

This rule illustrates the grammatical simplicity of a string clause combined with the `RUN` call to a Forth colon-defined word named `weight>2kg`. All clauses in an EXPERT-2 system would return a truth value. Thus, `animal has feathers` must eventually become known (to the inference engine) as either true or false. And the `weight>2kg` word must calculate the animal's weight (by whatever algorithm is captured in that Forth word), and return the result of a simple test: `2 >` .

The whole purpose of acquiring a composite truth value for the "IF" or antecedent side of a rule (also called the left-hand-side or LHS) is to determine if the consequent "THEN" (or right-hand-side or RHS) can be "fired." Thus, it really matters only as an implementation issue just how one designs the rule compiler. The EXPERT-4 upgrade of an EXPERT-2 rule lets the knowledge engineer precompile the strings with attached mnemonic names:

```
S: feathers " animal has feathers"
S: bird     " animal is a bird"
```

And, the rules are easier to type in:

```
IF feathers
ANDRUN weight>2kg
THEN bird
```

Of course, the inference engine developer is not constrained to a prefix environment; a postfix rule may be equally satisfactory:

```
feathers   IF
weight>2kg ANDRUN
bird       THEN
```

I present this example as an argument that, at this level of rule design development, it is a matter of personal preference. Steve Lewis (this issue) illustrates this point further.

Once one leaves the conversational approach to rule design, new vistas open. A typical OPS5 rule (after [BRO85]) for the monkey game looks like:

```
(p On::FLoor
   {(goal ^status active ^type on ^object-name floor)
                                  <goal>}
   {(monkey ^on <> floor)         <monkey>}
-->
  (write (crlf) (crlf) Jump onto the floor (crlf))
  (mofify <monkey> ^on floor)
  (modify <goal> ^status satisfied))
```

That same rule, written postfix in Expert-5 (versions below 5.7), reads:

```
R: ON-FLOOR
  GOAL NAME RECALL ['] FLOOR = .
  MONKEY ON RECALL ['] FLOOR <>
  --> CR." monkey jumps on floor "
    ['] FLOOR MONKEY ON POST
    GOAL-SATISFIED TRUE R;
```

In English, these rules read:

If the goal is to be on the floor
And the monkey is not on the floor
Then write a message that the monkey jumps on the floor
And modify the monkey's location to be on the floor
And mark the goal satisfied.

Further illustrations of this style of rule writing may be found in the FORPS article by Christopher Matheus (this issue).

But the rule design issue does not end the discussions we find. The Lewis inference engine uses linked lists as memory structures for locating objects, while the Matheus inference engine uses a table. EXPERT-5, for comparison, uses a variety of memory structures, including a Blackboard for posting and recalling the current states of the exercise implemented as an array, and rules compiled as colon definitions and collected into lists for execution. La Quey's technical note presents a reasoning system based on fuzzy sets.

Other forms of inference engine, such as those using statistical methods, are not presented here, but should be collected in a future issue. As well, rules are only one mechanism for encoding knowledge. Frames [MIN75] have become popular and are as easily implemented in a Forth environment as are rules.

Applications

The efforts of two medical researchers, both exploring brain- related issues, are discussed in this issue. Dana Redington and Robert Trelease both use personal computer-based inference engines in their sleep disorder research. The Redington work has focused on real-time analysis of the sensed environment, while the Trelease work has focused on off-line analysis.

A further application, though not implemented as an expert system, is the natural language processing program. The code presented in this issue represents a functional translation of the Dyer [DYE83] parser as an exploration of the implementation of a meaning-based parser. I intend the parser to be translated into a knowledge base in the EXPERT-5 environment to become a word-expert [RIE81] parser.

Conclusions

This issue presents aspects of many issues to the development of expert systems in the Forth environment. There remains a number of avenues for presentations of work in areas not covered by the papers collected here. There also remains a number of avenues not yet explored in the extensions of the work presented here. Let this issue, then, stand as a "call for papers." I am certain there are Forth-based projects, both at the implementation and the application level, that warrant collection and discussion. Let us fill in the gaps.

References

[BRO85] Brownston, Lee, R. Farrell, E. Kant, and N. Martin. *Programming Expert Systems in OPS5.* Addison Wesley, 1985.

[DYE83] Dyer, Michael. *In Depth Understanding.* MIT Press, 1983.

[FEY84] Feyock, Stefan. "Syntax Programming." *In Proceedings of the 1984 AAAI Conference.* American Association for Artificial Intelligence, 1984.

[MIN75] Minsky, Marvin. "A Framework for Representing Knowledge." In *The Psychology of Computer Vision.* Ed. P. Winston. McGraw-Hill, 1975.

[PAR84] Park, Jack. *Forth Expert System.* Mountain View Press, 1984.

[PER82] Perry, Michael. "Charles Moore's Basic Compiler Revisited." *Forth Dimensions* 3:6 1982.

[RIE81] Rieger, Chuck, and Steve Small. "Toward a Theory of Distributed Word Expert Natural Language Parsing." *IEEE SMC-11* **1**:43-51, 1981.