FORTH and ARTIFICIAL INTELLIGENCE

Robert La Quey

This paper is meant to be a brief report on work in progress. I will present opinions and some code. This is a first word and far from the last word. Thus I must first emit a disclaimer: I reserve the right to change my mind on any of these opinions and to completely rewrite any and all code at any time.

I have concentrated my efforts to date on extending FORTH to support the construction of Knowledge Based Expert Systems. I was first introduced to this subject by Jack Park and have benefitted greatly form an ongoing dialogue with him. Jack's work should be studied for an introduction to this subject. He wil also be a source of advanced work in this area.

My first effort was to implement a toy language which I called X (the language of eXpertise). The ideas in X are quite FORTHlike. An intermediate goal of this exercise is to find a minimal set of concepts to import into FORTH for the support of Artificial Intelligence.

I began by considering a Knowledge Base built of FACTs and RULEs describing relationships between FACTs. The first version of X assigned to each FACT a single string and a name. A FACT was also characterized by certain properties, to wit: The truth value of a FACT may be KNOWN or UNKNOWN. If KNOWN the truth value may be TRUE, FALSE, or FUZZY (more on FUZZY later). Jack Park provided a very useful crticism of the first version of X which led me to realize that a FACT can also be IRRELEVANT!, indeed this is often the class to which most FACTs belong.

RULEs tie FACTs together into logical systems. A rule in X took the form:

If FACT1 And FACT2 Or FACT3 ..... FACTL Then FACTM

where the logical precedence was established by scanning the expression from left to right and the logical operators were infix, a choice which made RELEVANCE easy to evaluate.

This is as good a place as any to explain RELEVANCE. Consider the previous RULE. If FACT1 is FALSE then the value of FACT2 is IRRELEVANT since the logical value of the phrase, FACT1 And FACT2, is already known to be FALSE. Similarly, if the phrase was FACT1 Or FACT2 with FACT1 being KNOWN TRUE then the truth value of FACT2 would not effect the outcome of logical evaluation of the phrase.

My approach then was to consider a FACT as rather like an extension of the notion of a TO VARIABLE, in rough outline:

: FACT CREATE LAYOUT DOES> EVALUATE ;

LAYOUT, the compile time procedure allots space in the dictionary for the truth value, the known flag and a pointer to antecedent FACTs when the FACT is the consequent of a RULE. A RULE when compiled patches up the antecedent pointer in the FACT, thus setting up the data structure which is used by the FORTH inner interpreter to perform backtracking.

At run time EVALUATE returns the truth value of the FACT on the FORTH parameter stack.

How it does this depends upon the KNOWN flag. If the FACT is KNOWN then the truth value of the FACT is simply fetched from its alloted location and placed on the stack.

If the FACT is not KNOWN i.e. the KNOWN flag is false then either the truth of the FACT is the consequence of some RULE, in which case we backtrack EVALUATing antecedent FACTs, or we must ASK the value of the FACT. ASK may query the operator or look the FACT up in a database depending upon the details of the system we have constructed.

Note that a FACT is just an ordinary FORTH word like a CONSTANT which produces a truth value on the stack. At run time EVALUATE may evoke quite a substantial exploration of the knowledge base as it works to determine the truth value of the FACT. Nonetheless the final result will simply be the return of a truth value. And since a FACT is just another FORTH word we can use it exactly like any other FORTH word. The FACTs thus provide a clean connection between ordinary FORTH and a Knowledge Base containg FACTs and RULEs.

We may thus use all of the Declaritive power of the Expert System approach to system building with out giving upany of the power associated with the classcial Procedural approach that we more often use in FORTH programming. I think that this gives us the best of both worlds, free from the dogma of either. But then I expect FORTH to provide freedom which is why I use it.

The ease with which we can fall back into FORTH has many implications. I wish to emphasize just one. Specifying an agenda for the evaluation of RULEs is almost trivial in this system. One can cause a system to try to evaluate any FACT by simply evoking that FACT as a FORTH word or as it is encountered as part of an ordinary colon definition. FACTs are EVALUATEd by bactracking but I alter the control flow explicitly and skip all over my Knowledge Base depending upon the outcome of FACT evaluation. It is thus extremely easy to implement mixed control strategies.

The introduction of fuzziness into the truth value causes surprisingly little complication. In classical set theory an element either belongs to a set (is TRUE = FFFF say) or does not belong to the set (is FALSE = 0 say). In fuzzy set theory, as developed by Zadeh and a host of others an element has a Degree of Membership in a set. TRUE and FALSE are just the limits. It turns out that many properties of Boolean Algebra generalize to take into account fuzziness.

For instance if one simply uses the definitions

: AND MIN ; : OR MAX ; and : NOT FFFF SWAP - ; (unsigned minus)

De Morgan's Theorem still works and ordinary Boolean Algebra with fuzzy truth values makes a kind of commonsense logic. Try it. I think you will like it.

FACTs can thus be made fuzzy by simply allowing them to take on values that are between TRUE and FALSE then suitably modifying the logical operators AND OR NOT as indicated.

I have only begun, however, to think through what happens if I allow the value of the KNOWN flag to be fuzzy. Comments would be appreciated.

In closing I would like to recommend the study of expert systems to my fellow FORTH programmers. They provide a refreshingly different way of viewing the art of programming. Expert systems seem likely to have signifigant commercial importance and are easily implemented in FORTH. My own long range goals are tending toward attempting to teach my computer how to read and write using a combination of Knowledge Based Expert System and Forth Technologies. It looks like a lot of fun! I may grow quite old trying.....

So it goes.

```
Scr # 9
  0 ( W                    10/7/85 rel )
  1  : PROGRAM CREATE 0 , 0 , 0 , ;                ( DOES> WHAT  ???? )
  2
  3  : VALUE       ( ^Fact --- Flag )             ;
  4  : KNOWN?      ( ^Fact --- Addr ) 2+  @       ;
  5  : ANT         ( ^Fact --- Addr ) 4 +         ; ( Ptr to Antecedent)
  6  : PRIMITIVE? ( ^Fact --- Flag ) 4 + @ 0=  ;
  7  : STRING      ( ^Fact --- Addr ) >FRONT @ ;
  8
  9  : T= ASCII T = ;  : F= ASCII F = ;
 10
 11  : TF? DUP T= IF DROP -1 -1 ELSE F= IF 0 -1 ELSE BEEP 0 THEN
 12    THEN ;
 13  : .FACT ASCII " EMIT STRING COUNT TYPE ASCII " EMIT SPACE ;
 14  : .PROMPT CR ." Enter T)RUE or F)alse for " DUP .FACT ;  -->
 15

Scr # 10
  0 ( W                    10/7/85 rel )
  1  : SET ( ^Fact,Val --- Val) SWAP 2DUP ! -1 SWAP 2+ ! ;
  2  : ASK .PROMPT BEGIN KEY TF? UNTIL SET ;
  3  : ANTECEDENT ANT @ >R ;
  4  : BACKTRACK DUP ANTECEDENT SET ;
  5
  6  : FINDOUT    DUP PRIMITIVE? IF ASK      ELSE BACKTRACK THEN ;
  7  : EVALUATE   DUP KNOWN?      IF VALUE @ ELSE FINDOUT   THEN ;
  8
  9  : FACT CREATE 0 , 0 , 0 , DOES> EVALUATE ;
 10
 11  : " ASCII " WORD DUP C@ 1+ ALLOT , ; IMMEDIATE -->
 12
 13  The layout of a fact is: Value,Knowflag,AntecedentPtr
 14  If Knowflag is 0 we must FINDOUT. If the AntecedentPtr is 0
 15  then ASK else BACKTRACK through the ANTECEDENTs EVALUATEing.

Scr # 11
  0 ( W                    10/7/85 rel )
  1
  2  : IN# QUERY BL WORD NUMBER ;
  3  : .PROMPT# CR ." Enter a number for " DUP .FACT ;
  4  : ASK# BEGIN .PROMPT# IN# NOT WHILE 2DROP REPEAT DROP SET ;
  5
  6  : CALCULATE ANT @ EXECUTE SET ;
  7
  8  : FINDOUT#  DUP PRIMITIVE? IF ASK# ELSE CALCULATE THEN ;
  9  : EVALUATE# DUP KNOWN?      IF @    ELSE FINDOUT#  THEN ;
 10
 11  : NUMBER CREATE 0 , 0 , 0 , DOES> EVALUATE# ;  -->
 12
 13    Note that if CALCULATE were BACKTRACK then the antecedent
 14    list could do the calcualting
 15

Scr # 12
  0 ( W Rule Compiler 10/7/85 rel )
```

```
Scr # 12
  0 ( W Rule Compiler 10/7/85 rel )
  1
  2  : RULES CREATE 0 , -2 , 0 , 0 ALIGN ] ;
  3  : ENDRULES DROP [COMPILE] [ ; IMMEDIATE
  4
  5  : APPEND, ( here,cfa --- cfa )
  6      SWAP HERE OVER - OVER 2- SWAP CMOVE -2 ALLOT ['] OR , ;
  7  : FIX_ANT_PTR    ( here,cfa --- cfa )
  8      SWAP OVER >BODY ANT ! ;
  9
 10  : If HERE ; IMMEDIATE
 11  : Then ( cfa,here --- cfa' )
 12     SWAP ' DUP ROT = IF APPEND, ELSE FIX_ANT_PTR THEN ['] EXIT ,
 13  ; IMMEDIATE    -->
 14   ( Rules with same consequent must be grouped together )
 15   ( APPEND, works by moving code back by 2 bytes          )

Scr # 13
  0 ( W                  10/7/85 rel )
  1
  2  : Or  DUP TRUE  = IF R> 2+ >R  ELSE DROP THEN ;
  3  : And DUP FALSE = IF R> 2+ >R  ELSE DROP THEN ;
  4
  5  : Not NOT ;        -->
  6
  7  The logical expressions are scanned from left to right. If the
  8  partial expression before an Or is TRUE then the next FACT is
  9  not relevant and is skipped over. If the partial expression
 10  before an And is FALSE then the next FACT is not relevant and
 11  is skipped over. The value on the stack at the end of the scan
 12  is the value of the total expression.
 13
 14
 15

Scr # 14
  0 ( X1                  6/5/85 rel )
  1
  2  : SHOW ' >BODY
  3       CR DUP .FACT ." is " DUP 2+ @
  4          IF   ." known " @ IF ." true. " ELSE ." false. " THEN
  5          ELSE ." unknown. " DROP
  6          THEN ;
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Scr # 15
  0 ( X1                  6/5/85 rel )
  1                              VARIABLE START
  2  : PROGRAM HERE START ! CREATE 0 , -3 , 0 , ;
  3  : (?) IF ." True " ELSE ." False " THEN ;
  4
  5  : WHAT? ;    ( Displays value of all facts )
  6
  7  : HOW? ;     ( Explains how a value is determined as in)
  8               ( "How do you know that?"                  )
  9
 10  : WHY ;      ( Explains the reasoning behind a question)
 11               ( as in, "Why did you ask me that?"        )
 12
 13  : ? ' DUP EXECUTE SWAP >BODY CR .FACT (?) ;
 14
 15   -->

Scr # 16
  0 ( Test FACTs & RULES  6/5/85 rel )
  1    : FLOOR ;
  2
  3    " aa " FACT A        " bb " FACT B        " cc " FACT C
  4    " dd " FACT D        " ee " FACT E        " ff " FACT F
  5    " gg " FACT G        " hh " FACT H        " ii " FACT I
  6    " jj " FACT J        " kk " FACT K        " ll " FACT L
  7
  8 RULES TEST
  9 If A And D Or H And I          Then C
 10 If C And D                     Then K
 11 If A Or  B                     Then J
 12 If K And J                     Then L
 13 ENDRULES
 14    : CEILING ;
 15

Scr # 17
  0 ( Animals   6/5/85 rel ) PROGRAM Animals
  1 " Animal has Feathers"  FACT FEAT " Animal lays Eggs" FACT EGG
  2 " Animal has Hair"      FACT HAIR " Animal eats Meat" FACT MEAT
  3 " Animal has Even Toes" FACT EVTO
  4 " Animal chews Cud"     FACT CUD
  5 " Animal has Long Legs" FACT LEGS
  6 " Animal has Long Neck" FACT NECK
  7 " Animal has Hair"      FACT HAIR
  8 " Animal gives Milk"    FACT MILK
  9 " Animal Flies"         FACT FLYS
 10 " Animal Flies Well"    FACT FLYW
 11 " Animal Swims"         FACT SWIM
 12 " Animal has Hooves"    FACT HOOF
 13 " Animal is Ungulate"   FACT UNG
 14 " Animal is Carnivore"  FACT CARN
 15 " Animal is Mammal"     FACT MAM       -->

Scr # 18
```