REvised REcursive AND? 'REPTIL :IS

Israel Urieli
Ohio University, Athens OHIO 45701

Background

Forth, described by some as the "hackers' language", is having a hard time being accepted within the computing community ranging from schoolchildren through computer scientists. Schools and colleges which are happily teaching BASIC, LOGO and Pascal, are not even considering Forth as an alternative. Probably the major reason for this is the singular lack of readability of Forth, demanding a strict self-discipline from the programmer with regard to style, documentation and structure. Many have addressed the question of readability (e.g. [1], [2], [3]), and have suggested that the major culprits include the cryptic symbology, the strange structured constructs and the lack of local parameters requiring an excessive dependence on stack operations. There have been many attempts to eliminate these problems, however, most of these have been in terms of extensions to the existing Forth kernel, rather than an appraisal of the language and some restructuring.

REPTIL represents an attempt to present a Forth-like language as a viable, pedagogically sound, alternative language for student programming at all levels [4]. Since its initial presentation in 1984 it has been totally revised and is currently being rewritten. This paper is a review of the infrastructure and some of the structured features of REPTIL.

Infrastructure

REPTIL has adopted the basic infrastructure of Buege's RTL [5]. RTL is a Token Threaded language having some unique features, including a one-to-one mapping between source code and object code. Thus no source code is ever retained. If source code is required for any purpose (such as editing) then it is regenerated from the lists of tokens in a "pretty-printed" structured format. This aspect is probably the most significant departure from Forth. The concept of source code screens and the associated virtual screen memory, which is fundamental to Forth, is no longer relevant. Other features of RTL which influenced the revision of REPTIL include its inherent recursive capability and the use of local variables.

Structures

The four fundamental structured constructs of REPTIL have been described previously [4]. In this paper we consider only extensions and modifications to two of these structures; the loop and the conditional branch. This is done by means of examples. Two good examples are taken from the source code for REPTIL's own system verbs. The operating system verb **RUN** is equivalent to the Forth verb **INTERPRET** and is defined as follows:

```
'RUN DO:
   R.RESET   \ initialize Return Stack pointer
   READ-LINE
   DO-LINE
   RUN
:END
```

This verb is defined recursively mainly to show how simple it is to use recursion in REPTIL. Note that the name (indicated by the quote) precedes the defining verb pair **DO: :END** as is common in many Forth-like languages such as STOIC, PISTOL and SPHERE.

In the coding for **DO-LINE** shown below we notice that the looping construct is given by the **REPEAT[ ]END** pair with optional exit conditions. Whenever the condition on the stack preceding an **?EXIT** is **TRUE**, then execution continues with the verbs following **]END**. This single loop structure can replace all the various loop structures in Forth. Soloway et al [6] reported extremely interesting empirical results which showed overwhelmingly that a loop form having an arbitrary exit condition allows the freedom of matching the programming strategy to a preferred cognitive strategy.

In the case of **DO-LINE** we see that when the end-of-line has been reached then the loop is exited. The body of the loop checks the following word in the input stream to determine if it is a **NAME?** (preceded by '), a **STRING?** (preceded by "), a **VERB?** or a **NUMBER?**, and the appropriate action is taken (**DO-NAME**, **DO-STRING**, **DO-VERB** or **DO-NUMBER** respectively), otherwise the system aborts with a call to **RUN**.

The coding for **DO-LINE** follows:

```
'DO-LINE DO:
  REPEAT[
    SKIP-BLANKS
    EOL?  \ end-of-line?
  ?EXIT
    NAME?
    ?THEN
      DO-NAME
    ?ELSEIF
    STRING?
    ?THEN
      DO-STRING
    ?ELSE
      BLANK
      WORD.0
      SCAN
      WORD.0
      VERB?
      ?THEN
        DO-VERB
      ?ELSEIF
      WORD.0
      NUMBER?
      ?THEN
        DO-NUMBER
      ?ELSE
        WORD.0
        UNKNOWN
        RUN
      ?END
    ?END
  ]END
:END
```

Notice the conditional branching structures including the **?THEN**, **?ELSEIF**, **?ELSE** and **?END** verbs. The Forth convention for conditional branch is inconsistent with a postfix world. Non-programmers of all ages were surveyed to determine which of the following two examples is the more meaningful:

1. **HUNGRY? IF EAT ELSE STARVE THEN**
2. **HUNGRY? ?THEN EAT ?ELSE STARVE ?END**

In all cases the subjects found the first statement difficult to read or ambiguous and contradictory (if one eats, then why should one be hungry?), and the second statement perfectly clear.

In REPTIL the conditional branching structure also includes optional **?ELSEIF ?THEN** pairs. These are widely used in procedural languages such as FORTRAN or Pascal as a more readable alternative to deeply nested IF structures, however they have not been used in Forth-like languages.

Closing Remarks

REPTIL is still in the development stage and significant changes may occur before it is released. The main effort is being directed to a fundamental readable (and writeable) syntax in order that REPTIL can be a viable alternative to BASIC or LOGO as a student programming language at all levels. The reason for this continued premature exposure is to obtain feedback and criticism from the Forth community.

References

1. H. Glass, "Towards a more writeable Forth syntax", *Proceedings of the 1983 Rochester Forth Conference, June 1983.*
2. E. E. Bergmann, "PISTOL - a Forth-like Portably Implemented Stack Oriented Language", *Dr. Dobbs Journal, Number 76, February 1983.*
3. C. B. Duff, N. D. Iverson, "Forth meets Smalltalk", *The Journal of Forth Application and Research, Volume 2, Number 3, 1984.*
4. I. Urieli, "HELLO, A REPTIL I AM", *Proceedings of the 1984 Rochester Forth Conference, June 1984.*
5. B. Buege, "Status Threaded Code", *Proceedings of the 1984 Rochester Forth Conference, June 1984.*
6. E. Soloway, J. Bonar, K. Ehrlich, "Cognitive Strategies and Looping Constructs: An Empirical Study", *Communications of the ACM, Volume 26, Number 11, November 1983.*