Letters to the Editor

Forth and Standards

For the last ten years I have been a silent witness to the several attempts at establishing a set of standards for the Forth language. I have greeted the widening of this effort through the involvement of ANSI as a trend towards full recognition of Forth as an academic and professional system. Recent reports on the work of the X3J14 ANS Forth Technical Committee (for example, see [DUN88]), however, have made me concerned that the ongoing standardization smacks of Medieval Scholasticism. (Instead of arguing about angels on a pin, members of the standards team argue about whether words should be prefixed by a **D** or a **2.**) Lexical arguments[1] seem to take more time than semantic concerns such as machine-independent description of the Forth interpreters,[2] language-extension mechanisms,[3] as well as address-width problems. What seems to have escaped the notice of all standards committees is that Forth is unlike all other conventional programming languages!

A conventional programming language may be likened to a child's Tinker Toy™ set. Here, the sizes and colors of the components — wheels, rods, and connectors — are fixed and it is not possible to use a new item, say, a cube, because it is neither provided nor constructible from the primitive elements. A child compiles a structure and then plays with it — compile time strictly precedes run time. The semantics of such a language can be specified by means of a single monolithic interpreter which works according to the fetch-decode-execute cycle.

Forth, on the other hand, may be likened to a block of wood and a sharp whittling tool. Each element of a structure being built must be individually constructed, so that arbitrary extensions of the base language are possible. The Forth user constructs small components that can be individually optimized to fit the task in hand. Not only is the abstract interpreter distributed over the various data types,[4] but compilation and execution can be arbitrarily interleaved.

Can we ever achieve an adequate level of standardization in view of the high mutability of Forth? I believe that we can, but we must proceed in an unconventional manner: The first step is the identification of an irreducible subset of Forth. This subset is the smallest Forth that can compile itself and be capable of extending itself to a more useful system. As a first proposal, may I suggest the following:

1. Inner interpreter — **NEXT**, **NEST**, and **UNNEST**
2. Outer interpreter — **INTERPRET**, **EXECUTE**, **ASSEMBLE**, **COMPILE**, **CREATE**, **DOES>**, and **;CODE**, and
3. Only other words are those needed to define the above ten words.

---

[1] I suppose one should be thankful that Forth is a syntax-free language!

[2] Some work in this direction was reported in [SOL82], [SOL83], and [SOL84].

[3] A comprehensive survey of the extensible languages of the 1960s can be found in [SOL74].

[4] Forth data objects built by means of the **CREATE...DOES>** mechanism are active objects, by contrast with data in conventional languages which are strictly passive.

Formal definition of the minimal subset would give us confidence that programs could be written to be correct. Formal definition of the extension mechanism would ensure every step of the way from the minimal subset to the final product could be made correctly. Once the theoretical basis of Forth is made secure by the above approach, we can proceed to the problem of optimizing the performance of the extended Forth system. It is possible that this process will lead to separate extended Forths optimized for numerical analysis, text processing, database management, etc., but this should not be taken as a calamity — after all, are we all not happy with the concept that Forth becomes the application?

In conclusion, let me summarize my theses. *Standardization of Forth should only encompass the minimal subset containing no more than the words needed to auto-compile a Forth system.* In concert with this work, the formal definition of Forth, especially, the extension mechanism, should be undertaken to provide the theoretical foundation for the total definition of the language.

Nicholas Solntseff
McMaster University
Dept. of Computer Science and Systems
1280 Main Street West
Hamilton, Ontario, Canada L8S 4K1

References

[DUN88] R. Duncan, "ANS Forth Meeting Notes," *Forth Dimensions*, **X** (Number 1, May/June 1988), pp. 24–25.

[SOL74] N. Solntseff and A. Yezerski, "A Survey of Extensible Programming Languages," *Annual Review in Automatic Programming*, Vol. 7, Pergamon Press, London (1974), pp. 267–307.

[SOL82] N. Solntseff, "An Abstract Machine for the Forth System," *Proceedings of the 1982 Forth Conference* (1982), pp. 149–155.

[SOL83] N. Solntseff, "An Instruction-set Architecture for Abstract Forth Machines," *Proceedings of the 1983 Forth Conference* (1983), pp. 175–183.

[SOL84] N. Solntseff and J.W. Russell, "An Approach to a Machine-independent Forth Model," *Proceedings of the 1984 Forth Conference* (1984), pp. 121–139.