Request for comments
(appendix to workshop introduction)

Andrew Read
andrew81244@outlook.com
July 2016

An Axiomatic Approach to Forth

1. Introduction

Forth has traditionally been implemented by writing a certain number of code words in assembly language, out of which the remainder of the Forth dictionary is built up. Forth virtual machines may implement code words in C or another high level language. Forth processors may offer instructions that correspond directly to code words.

Traditionally there have been few constraints on code words. ANSI Forth defines the behaviour of high level Forth words and leave implementation details to the system designer. This approach has its benefits, but also leads to certain practical problems.

Firstly, mature Forth systems that are ANSI compliant may actually behave differently, especially when programmed at a “technical” level.

Secondly, implementers of new FORTH systems, virtual machines or Forth processors have to “make it up from scratch” every time. Whether the resulting Forth systems are truly ANSI compliant cannot be tested until after completion.

Thirdly, there is no straightforward way of porting mature Forth implementations to new targets. Each time new, machine specific, code words must be written and somehow tested before the porting of Forth itself can begin.

2. A conceptual viewpoint

A conceptual objection may also be made: Forth has been long proven to “work” as a programming language, but because the code words upon which the implementation of every Forth system depends are arbitrary, there is not a certain foundation to the language.

On the other hand, there is an opportunity here: Forth has no syntax so the behaviour of Forth words can be completely defined in terms of their effects, irrespective of the context in which they occur. So it should be possible to determine a completely deductive chain of logic from the most fundamental underlying elements of Forth through to ANSI Forth words, via a well-defined set of code words.

3. Objectives of the project

This project aims to take a deductive approach to the definition of Forth from conceptual underpinnings. There are four stages

(A) Elemental structures (“structures”)

Identify and document the elemental structures of Forth at a conceptual level.

“Structures” in this context means something akin to “physical entity”, or perhaps “mechanical entity”, rather than just data structures in the traditional sense. Hopefully the intended meaning may become clearer through the following discussion.

Some structures are explicit in Forth (e.g. the parameter stack), while others are implicit (e.g. the program counter, system memory, or the locus of arithmetic logic). Yet others may require more careful thought. For example, is the return stack a single elemental structure or is it actually the mapping of multiple conceptual elements (a LIFO store accessed with >R and R> and a subroutine return program counter store) to a single implementation entity? What kind of structure is the Forth dictionary itself?

The objective of this stage will be to “find” all of the elemental structures that underlie what we commonly understand as Forth, to properly separate them, and to describe them concisely and rigorously.

(B) Elemental operators (“operators”)

Identify the elemental operators which act on the structures and document them by stating their effects.

Again some elemental operators make themselves very evident and in fact are cognates with Forth words. For example, “+” is an operator that acts on the parameter stack, the locus of arithmetic logic, and on the parameter stack again.

Other operators are less obvious, for example is there an operator “BNE”, that acts on the program counter conditionally depending on the value held at the top of the parameter stack?

The objective of this stage will be to find all of the elemental operators that we believe comprise Forth and document them in the form of a table that shows their impact on the elemental structures.

This stage is likely to be highly iterative with the identification of elemental structures. For example, when we consider “BNE”, if it

is an operator, does it imply there is also a structure that is the locus of logical comparison?

Referring to the title of this RFC, the elemental structures and operators might loosely be considered a set of “axioms” for Forth.

(C) Code words

The next stage of the project is to bring together the structures and the operators into a suitable set of Forth words from which a complete Forth implementation can be developed.

The code words serve as the abstraction layer to provide “Forth-like” access to the elemental structures and operators.

Some code words may map directly onto individual operators (perhaps “+” for example). Others code words will be combinations of operators, acting serially or in parallel.

The code words will need to take account that there may be differences between the data width of the Forth system (e.g. 32 bits) and the data width of the underlying structures (e.g. 16 bits or 8 bits). This project does not intend to prescribe any expected data width at either the structure or the Forth system level.

From a practical perspective, code words may be implemented in a machine-dependant manner in the language of the underlying system, as has always been the case. (That language might be C for a Forth virtual machine, assembly language, or the primitives of a Forth processor.) However, the implementation of the code words will no longer be arbitrary because (i) the set of code words will be explicitly defined and (ii) the function of each code word will be completely specified in term of the fundamental structures and operators.

This stage of the project is likely to be rather judgmental. The optimally chosen set of code words is unlikely to be the minimal set (for example there is actually no need of “+”, provided we have “0” and “-”, but is this a sensible economy?).

A staging post of this phase in the project is likely to be the articulation of a set of policies or guidelines for deciding which words should be defined in terms of the fundamental structures and operators (the code words) and which in terms of other Forth words (the remainder of the dictionary).

(D) Forth implementation

Finally, the code words can be leveraged to develop an ANSI Forth implementation.

4. Working approach

(A) Relative weighting of effort

I anticipate that the first two stages, finding the elemental structures and operators likely represents 60% of the effort that would be required. Although a first draft can no doubt be drawn up quickly, consideration of subtle points and generally iterating and polishing the thinking will take much more time. The third stage, the code worlds is perhaps 25% of the effort, and much of that spent on consideration of words at the boundary between “the code” and the rest of the dictionary. The final ANSI Forth implementation, whilst probably the greatest number of written lines, may only be 15% of the effort if the Forth implementation is limited to the CORE wordset and a few others, and good advantage is taken of readily available prior work.

(B) Verification

It will be necessary to verify the results of each stage. A number of possible approaches exist and the actual verification approach adopted will depend on the preference of the project participants.

Firstly, there is the possibility of using some sort of “logical calculus” to prove results in a manner similar to pure mathematics. Although this approach has been adopted before, particularly in relation to verifying stack operations, experience suggests that such an approach is likely to prove unwieldy in practice and that the difficulty of developing the “calculus” in the first place will probably exceed its benefit.

Secondly, there is the use of informed debate to discuss critical decisions, not just in terms of functionality but also from the perspectives of desirable aesthetics and symmetry. We can call this the “philosophical” approach. *Hoc tam ars quam scientia est*. Examination from an aesthetic perspective will be invaluable to for an elegant result.

Thirdly, there is the mechanical approach. By explicitly simulating (perhaps with pencil and paper at first) the structures, operators and code words it should be possible to verify the effect of any sequence of operations. The mechanical approach needs to be alert to “corner cases”, and here again there is a role for informed debate as a source of suitable challenge.

Finally, and this is really an extension of the mechanical approach, a working Forth system built on these foundations will help to convince that the foundations are satisfactory.

5. Uses and benefits

It is intended that the four components that will be developed in this project (the structures, operator, code words and ANSI Forth implementation) may serve as an “axiomatic” reference model that enhances and clarifies the Forth language. It is not intended that they should be advocated as “standard”, or that they should proscribe other approaches. If the reference model is intellectually appealing and helpful in itself, that will be justification enough for the effort expended.

The root of my own interest in this project is my experience of developing an instruction set and Forth system for the N.I.G.E. Machine. In the last few years I have become interested in how Forth constructs can be visualized as structures and then taken from software into hardware. This approach has allowed exception handling and multitasking to be implemented as atomic machine language instructions in the N.I.G.E. Machine.

Ulli Hoffmann mentioned to me some time ago how a Forth meta-compiler could be used to “make seamless” the Forth held in RAM and that included from source files on SD-card. I now wish to extend the Forth system software and before doing so it would be expedient to migrate the N.I.G.E. Machine to a meta-compiled system. At the same time, I would like to re-examine and potentially reconfigure the N.I.G.E. Machine instruction set. Both of these aims will be better accomplished in the light of a conceptually rigorous approach to the fundamental structure of Forth. Hence my wish for a reference model with axiomatic foundations.

I believe the reference model could also be interesting to anyone working with Forth virtual machines, since there is really very little difference between a Forth processor in hardware and a Forth virtual machine in software.

The reference model might be helpful to anyone who wishes to use Forth on the multitude of new microprocessor-based development boards since consistent system behaviour will be assured. In addition, perennial practical difficulties such as efficient Forth file transfer can potentially be addressed at a low level by defining interfaces at the level of elemental Forth structures and building suitable operators for their handling deep into the language.