RFC

Andrew Read (AR) / Ulli Hoffmann (UH)
EuroForth 2017

Forth: A New Synthesis

1. Introduction and objective

It is well-known that Forth scales-down well to the smallest platforms and applications. However, it is less obvious that Forth scales-up well to large applications or development projects. Our hypothesis is that some of the features of Forth that enable it to minimize so successfully, are constraints on the language in scaling up.

These days small code size and fast execution speed are relative rather than absolute merits. This project aims to produce a new synthesis of Forth that rebalances the requirement for scaling-down against the opportunity for scaling up.

We propose a “new synthesis” of Forth in a similar spirit to the Forth Modification Laboratory workshops.

Any new synthesis of a computer language runs the risk of being formed solely out of individual preferences and experiences. That remains the case with this project, but to provide guidance two governing principles have been adopted: biological analogy and disaggregation.

2. Principle: biological analogy

The biological cell provides a fascinating model of a complex system that processes stored information. With imagination, direct analogies can be drawn between the component parts of a cell and the component parts of a Forth system (Appendix I)

The most compelling rationale for consulting our understanding of the biological cell for ideas is that the biological cell is a proven-successful system that scales-down (to virus and single-celled life-forms), scales-up (to self-conscious Homo sapiens), and diversifies (to the different kinds of cell specialization within a single organism and to the multitude of life-forms on Earth), whilst broadly maintaining a common internal architecture of structure and function.

3. Principle: disaggregation

The project aims to identify the component parts of Forth and separate them, even when this separation may be to the detriment of efficiency. Experience suggests that proper disaggregation by itself frequently solves existing problems. Table 1 makes some specific proposals

	Traditional Forth	**New synthesis**	**Implications**
1	Forth words have direct access to byte-by-byte contiguous memory	Interpose an allocation based memory management system between system memory and the Forth system	Allows further disaggregation of the dictionary and heap (e.g. S" or colon data) data structures
2	Dictionary and the heap may be built up together in memory	Impose the separation of the dictionary and the heap	The dictionary and the heap may independently be rewound or modified
3	Dictionary may contain headers and code	Use the allocable memory mode to separate headers and code	All Forth words can be erased and rewritten
4	The input stream is locked into the INTERPRET loop	Interpose an extendible text processor that exclusively handles the input stream	Rather than using parsing words, the text processor provides a general model for extending the language
5	Both interpret or compile states are handled by the INTERPRET loop and state-smart words	Disaggregate by eliminating state: everything is compiled, but depending on context some compilations may immediately be executed and then rewound	Removes ambiguity in word definitions and divergence in the way the language is extended

Table 1. New synthesis disaggregation proposals

4. Practical experiments and observations

UH prepared two model systems for experimentation since EuroFORTH 2016

i. A Forth system built on the GO language that implements an allocable memory model foundation for a Forth system. The reason for wanting this memory model was to enable the redefinition of dictionary Forth words with new code. We experimented along two dimensions, as follows

What to do with old instances of the word? / what to do with recursive references?	**Not true recursion - reference the prior word definition, if available**	**True recursion – reference the new definition**
Old instances retain old code	Traditional Forth	“Mixed”
Replace all instances with the new definition	n/a	LISP-like

One interesting implication of the LISP-like model is that it might allow a Forth system to completely replace its dictionary with a new set of definitions. This is explored further in future directions, below.

ii. An extension to Forth that “opens” the INTERPRET loop and interposes an extendible text processor. The key finding was that with this model there is no need for recognizers as their function is already a natural part of the system. See "Recognizers Dissolved", Ulli Hoffmann, EuroForth 2017

5. Conceptual next steps

We are interested in developing the new synthesis, guided at a high level by the biological analogy. Some specific thoughts are as follows

i. All living organisms on the Earth share a common (i.e. highly evolutionary preserved) fundamental core (DNA/RNA, the genetic code, proteins, cells, etc.). Yet the complexity and diversity of life on Earth depends on the variation between their biological systems. For example, the fore limb apparatus can be adapted into arms and hands by primates, into legs by four-legged animals, or into wings by birds.

Should the same be true for Forth systems? A common operating framework (WORDS, STACKS and BLOCKS?) with a minimal dictionary in

all Forth systems, but then wide variation at higher levels between systems in both the vocabulary of available words and their definitions. This would be contrary to the ANSI approach, but could it be sympathetic to Chuck Moore's original vision?

ii. All multi-cellular organisms grow from a single cell, typically the egg.

Should a similar approach be taken to 'growing' Forth onto a new target? For example: The Forth 'egg' in ROM implements the minimal framework and dictionary. It has the capability to read an input-stream from a small EEPROM, but otherwise no input/output firmware. The EEPROM contains plain-text Forth code in byte-by-byte format. The egg reads from the EEPROM and 'grows' by implementing further i/o and other firmware, including if necessary a FAT file system on SD card. An SD card may contain further Forth files with application software. At each stage the Forth system will be re-engineering its input stream apparatus with a more sophisticated version.

iii. Biological systems are not static – they continue to evolve.

Should Forth be the same? For example, rather than standardizing more and more of the language, should the curation of Forth encourage "forking" of new versions of Forth, even if most die out.

iv. Some biological systems are super-organisms, such as colonies of ants or bees. Individual organisms are specialized into different roles, and the loss of one individual does not jeopardize the hive.

Should some Forth systems strive to adopt a distributed model, with a "queen" system spinning off specialized, limited capability "worker" Forth systems on very low cost peripheral processors.

v. Biological organisms are single-purpose: a dog is a dog, a cat is a cat and a man is a man. Conventional computer systems are now multipurpose: a desktop computer is a word-processor and a CAD platform and a music player. This results in the "layered" operating system / middle-ware / package approach with standardized interfaces, all programmed by thousands of different individuals

Is Forth better off devoting itself to single-purpose systems that are developed by individuals or small teams? In this domain inter-compatibility, wide library support, the layers and standardized definitions are not critical. (See also unikernal.org for some interesting thinking on this topic.)

In this model, ANSI Forth remains important for thought leadership and communication, but not for its word definitions *per se*.

In exploring these ideas we will certainly need to address at least two questions.

Firstly, what is the basic operating framework needed across all Forth systems, analogous to the highly conserved structures of the biological cell. For example, memory allocation, a dictionary, a heap? Specified at what level of detail?

Secondly, what should a Forth ‘egg’ contain? In common with a biological egg it should contain the operating framework and a small dictionary. What needs to be in the dictionary? How is the egg ‘primed’ at power on, etc..

6. Conclusion

We are exploring the future of Forth in the spirit of the Forth Modification Laboratory. Our work is motivated by the observation that Forth has been left behind as computer systems have scaled up, and by our optimism that somehow Forth might still be refreshed and reinvented in interesting new ways.

Two principles are guiding us. Firstly, on modern hardware minimalization of code size and optimization of execution speed are no longer extreme necessities. We are prepared to make some sacrifices on these dimensions for the sake of a more disaggregated architecture that has the potential to scale up more easily. In other words we want to be more aware of the degrees of freedom on any given target and make conscious decisions that may differ from historical precedent.

Secondly, we note an exciting analogy between Forth and biological systems and wish to see if this inspiration can guide us to a “Cambrian Explosion” in the diversity and sophistication of Forth Life.

Appendix I – Forth biological analogy

	Cell	**Forth**	**Analogy**
1	DNA is the genetic material that defines the functioning of the cell	Source code is the material that defines the functioning of the application	DNA <=> Source code
2	The DNA of a cell is located in the chromosomes	The source code of a Forth application is located in blocks	Chromosome <=> Blocks
3	Foreign DNA may be expressed in a cell (viruses / genetic engineering)	The input stream may be directed to the keyboard, serial line or other port	Foreign DNA <=> Keyboard input
4	DNA that is to be expressed is translated into mRNA	Source code that is to be run is directed to the input stream	mRNA <=> Input stream
5	Gene expression is mediated by controlling the transcription of DNA into mRNA	Source code may be chosen by directing the input stream to the relevant blocks	Gene expression control <=> Redirection of the input stream
6	mRNA is translated into proteins; proteins make the cell function	Source code is compiled into words; words make the application function	Proteins <=> Words
7	Proteins are comprised of amino acids	Words are comprised of assembly language instructions	Amino acids <=> assembly language instructions
8	Translation takes place at the active sites of ribosomes	Source code is compiled into words by the compiler	Ribosomes <=> Compiler
9	Translation is mediated by tRNA molecules that parse the sequences of the genetic code	Compilation is mediated by recognizers that parse the input stream	tRNA <=> Recognizers
10	mRNA coding regions begin with start codons and end with stop codons	The input stream defines words with colon and semi-colon	Start codon <=> Colon Stop codon <=> Semi-colon