Perspective MetaFORTHness

Mikhail Kolodin
St.Petersburg, Russia
`http://myke.da.ru, myke@mail.ru, 2:5030/74.42`

1999-09-17

Abstract

`FORTH` is known to be good for many reasons, but for the person in science it should be primarily rewarded as a *meta*-system capable of self-development, self-modification, being a *meta*-transitional tool.

At least it was so by now. The `FORTH` implementations appearing in the last days enrich `FORTH` with new mechanisms, technologies, constructions, but they seem to lose the initial flexibility. The requirements for and directions of meta-ways of `FORTH` development are discussed in the paper.

`FORTH` is well-known *meta*-instrument [1, 3, 6, 7], capable of self-reproducing and self-tuning. The technique of *meta*-compiling is traditionally used for that. With it the `FORTH` system takes the complete own text on input, compiles it in the special context, and writes the result as the new kernel.

The `FORTH` systems appearing in the last time use advantages of the operating systems they are used in. E.g. `FORTH`s that are complied for MS Windows use WINAPI interface. Since the access to such interface functions is already provided by existing high level languages implementations, say, MS Visual C++, they use C++ functions in the lower (kernel) level and, in fact, are themselves written in C++.

But it means that `FORTH` is no longer written in `FORTH`; it cannot be recompiled, reassembled using only its own facilities, but requires external C++ compiler and maker (linker). This happens in many cases nowadays. It cannot be considered a good practice.

Let us mark the most important features of traditional *meta*-`FORTH` implementation that make it so interesting in *meta* plan [2, 3, 4, 5]:

- it has the complete text of itself in self-processable form (maybe using built-in assembler);

- it has a special context (created in FORTH itself) that allows the FORTH system be compiled out to separate space that can be dumped or saved in some other way;
- it has special words to facilitate the setup and the process of *meta*-compiling, so the program does not have to be changed significantly when used in normal or *meta*-mode;
- it does not use external tools to be recompiled; everything needed is contained within the same system, usually in loadable external files (blocks).

From the point of view based on these definitions we cannot consider modern FORTH implementations to be *meta*-systems; though, according to some new research, the term *meta* itself has changed: we now deal with complete and partial *meta*-systems, open and closed *meta*-systems:

complete *meta*-systems: containing all the necessary parts and sources within them;

partial *meta*-systems: only some subsystem, treated as *meta*, is *meta*-processed, while other may be processed with external tools;

open *meta*-systems: allowing access to all its parts (subsystems or external tools);

closed *meta*-systems: some parts (subsystems) are unavailable and are processed automatically, by their own means (as grey or black boxes).

With these refined definitions we now distinguish

- Linux as a complete open *meta*-system when compiled totally,
- but as a partial closed *meta*-system when tuned in or recompiled partially;
- Beta-FORTH as complete open *meta*-system, when recompiled using its own *meta*-package;
- SP-FORTH and WIN32FORTH as complete closed *meta*-systems when compiled as usual.

Note that some basic level is required to be fixed, usually at the OS level: we shall not always require recompiling from the hardware level.

Nevertheless, though we redefined *meta*-system in a pleasant way, we should reconsider our understanding of FORTH strategy:

- FORTH implementation should be completely expressed in terms of FORTH itself;
- FORTH should be *meta*-processed transparently for the user in respect to the real tools used in the process;
- as many as possible subsystems and external systems, used by FORTH, should be directly accessed from FORTH.

This will require additional theoretical investigation and practical implementation and testing, but it has good perspectives.

Having this in mind we shall succeed in building *true meta*-FORTH systems.

References

[1] Baranoff S. N., Nozdrunov N. R. The FORTH language and its implementations. — Leningrad: Mashinostroyeniye, 1988. — 157 p.

[2] Baranoff S. N., Kolodin M. Y. The FORTH phenomenon. // System Informatics, #4. // Novosibirsk: VO Nauka, Sibirskaya Publishing Company, 1995. — Pp. 193–271.

[3] Kolodin M. Meta-technology: Purpose and Implementation. (In Russian) //Information technologies and intellectual methods. — SPb: SPIIRAS, 1995. — Pp. 83–86.

[4] Kolodin M. Preprocessing and Macroprocessing in FORTH-Based Meta-System. (In Russian) //Information technologies and intellectual methods. — SPb: SPIIRAS, 1995. — Pp. 86–93.

[5] Kolodin M. Extended Possibilities of Decompiling in FORTH. (In Russian) //Information technologies and intellectual methods. — SPb: SPIIRAS, 1995. — Pp. 93–100.

[6] Kolodin M. Meta-systems in Scientific Research. (In Russian) //Paper for International conference "Regional informatics-96" (1996, St.Petersburg).

[7] Kolodin M. Meta-features in FORTH. (In English) //Paper for International conference "EuroForth-96" (1996, St.Petersburg).