Demise of the Metacompiler in cmForth

Jay Melvin, RPCV

Abstract

Metacompilation is a technique of regenerating a computer's executive program by compiling itself to provide a new system. Traditional metacompilers manipulate memory space operators to distinguish between native, host and target memory so to enable use of much of the same source code for each space. The functions of the metacompiler are described. An innovation and simplification of Forth is shown to rest on a new implementation of the metacompiler's memory manager. A Glossary of terms is provided.

Introduction

A metacompiler **(1)** is a program that converts the human written source description for a Forth system into machine executable object code. Since the Forth system's source can be so recompiled, modifications and enhancements are readily made and the system is amenable to customization at any level.

Metacompilation Details

Meta compilation is used to solve several problems that regularly arise in the course of designing application solutions. Usually programming tools are general in purpose and may need honing for a particular endeavor. Program customization results from changing environments, e.g., disk drive controllers, ROM or RAM adjustments, different processors, repairing bugs or adding new functions such as an operating system or language. Such pronounced differences in application are easily accommodated by building a new program with the metacompiler. Note that the source code that compiles into the executive (nucleus) is mostly the same for the native, host and target programs; any differences result from the systemic changes just mentioned.

The native Forth is the system used to design, develop and debug applications. When an application's solution entails another Forth - a new development environment - then the metacompiler is loaded to provide host dictionary entries that can be searched and otherwise used to build a target image in the host' memory. Host's memory is that native Forth's memory occupied by the metacompiler. Eventually, the host's target image is installed in the target computer where it will run; the target computer may be the same hardware as the native uses but, at target run-time, this new system will be in place.

In traditional Forth programs **(2)**, memory management of metacompilation involves:

A. Memory operators including comma, fetch, store, move and dump as well as various compilers, the assembler and flow-control directives separately defined to work in native, host and target memory spaces.

B. Appropriate versions of **CREATE** for native, host and target spaces for:

1. Compiling name fields for interactive interpretation to provide feedback.

2. Linking each word to the previous word for ROM/RAM separation, interactive interpretation and incremental compilation.

3. Compiling "locate" pointers for editing-compiling interactively.

4. Compiling appropriate run-time actions for each kind of word, i.e., variables differ from constants and arrays and so on.

5. Calculating target addressess for the yet unfinished program.

Given a running native Forth, the metacompiler is loaded so to provide these host memory management services. Discussion of the compiler's extender, **DOES>**, its temporal actions and its compilation, is beyond this article. The target's nucleus is laid down into memory followed by the metacompiler's compiler which lays down the target's own compiler and the target's application. Programs to be run at different times are maintained in different areas of memory. In traditional Forths, the memory manager that controls where target words are laid down is **T**, "to target", which calculates the target dictionary's offset from that of the host. All the operators and **CREATE** are redefined in the metacompiler using **T**. Substantial source code is required and understanding it takes some commitment in study.

Metacompilation in the Forth Engine

The invention of the Forth Engine required rethinking of the virtual machine. Hindsight allows us to share Chuck Moore's insight **(3)** that all the functions of memory management, mentioned above, can be performed at another level with a great reduction in complexity! cmFORTH replaces the metacompiler program with the single word **{**, "switch". In cmFORTH, the entire metacompiler is reduced to the following:

```
VARIABLE H' 1000 ,
   : {   dA @  HERE  H' 2@ H !  dA !  H' 2! ;
   : }   { ;
```

The switch allows the native compiler to generate host and target code while maintaining the virtue of reusing the original source code wherever possible. The double variable **H'** points to the target dictionary and contains the relocation offset. **dA** is the variable containing the "delta Address", i.e., the difference between the host and target dictionary pointers **H** and **H'**. The two dictionary pointers are switched whenever **{** or **}** are invoked; they are switched to control space and compilation happens as usual in the native version, without new memory operators or another **CREATE** for the host or the target. The relocation of the target object is choosen to simplify usage of tools such as **DUMP**. Examining the target image's new version of,

cont'd next page

say **DUP** , with the native's **DUMP** requires only adding the offset to the native version's address. Differences in the programs are made conspicuous by a digit in the address.

The switch performs all meta compilation memory management functions, except for **DOES>**, by merely redirecting compilers and tools to appropriate memory space associated with the three programs, native, host and target. The metacompiler is thus diminished in Forth's spirit of parsimony and rendered in one line of source!

Notes

(1) Such as the target compiler designed by Dean Sanderson (1977-8) at FORTH, Inc. "Target compiler" is a trademark of FORTH, Inc.

(2) FORTH Inc.'s polyFORTH, Laxen & Perry's F83 or Zimmer, et al., F-PC are familiar examples.

(3) cmFORTH was the first (1986) programming environment for the Forth Engine.

Glossary

WORD a named collection of machine instructions; the name suggests function.

COMPILER WORD(s) that lay machine instructions into memory for future use.

PROGRAM a WORD that provides some service to the user, e.g., development environments, operating systems and languages are programs.

NATIVE computer hardware and software used for application development.

TARGET computer hardware and software under construction for future use.

HOST computer hardware and software used to construct TARGET; runs both its native PROGRAM and the metacompiler.

NUCLEUS real time executive, stack manipulators, arithmetic and logic operators.

INTERPRETER a program that executes WORDs or stacks numbers; input is under program control either from the keyboard, disk or some other I/O port.

DICTIONARY program memory organized as linked WORDs.

CREATE a key word needed to compile every WORD; it links to the previous WORD in the VOCABULARY.

LINKER every incremental compilation provides connection of the new routine to extant WORDs in the dictionary; one of the functions of CREATE .

RUN TIME when the words will perform application duties, "tomorrow".

COMPILE TIME when the word is built in the native system, "now" ; distinct from the time when the native itself was built, "yesterday".

RUN or **EXECUTE** implies that the interpreter has been pointed to some memory space where it finds WORDs that force the machine to proscribed action.

LOAD or **COMPILE** or "lay down into memory" refers to the act of programming under an interpreter as opposed to programming under an editor.

Jay Melvin writes code for Tracor Ultron Labs to run on the RTX processor; he also worked with Forth at MAXTOR Corp. and FORTH,Inc. He wrote the shadow documentation for the first release of Computer Cowboy's cmFORTH. He is a Returned Peace Corps Volunteer.

❑ ❑ ❑

FORTHought

Chuck Moore

Computer Cowboys

Recently I've been doing something really odd. I've been using someone else's software. Takes me right back to the preForth days. You see, I want to design another chip. This means playing by the rules. At present that means workstation, Pascal program (I think) and UNIX. An opportunity to get an update on the state-of-the-art.

When Bob Murphy and I did the Novix chip, Bob ran the Highland system he knew so well. I couldn't hack the slow, clumsy interaction and concentrated on simulating in Forth. This time I'm in the hot seat and have acquired the wisdom and patience to cope with the system. I was curious and apprehensive about how far out of touch I had gotten after spending twenty years with Forth. Imagine my relief in learning that nothing had changed.

We used to use mainframes whereas we now use a workstation. Displays were rare and are now in color. Operating system was OS, now UNIX. Language was Fortran, now Pascal (C?). But nothing has changed. I think we could have predicted 1989 very well in 1969. After all, where there are no problems there is no need for change.

I'm using a Sun-4 workstation, 8 Mbyte RAM and a 370 Mbyte disk backed by Ethernet. With the SPARC micro, this is surely current hardware. It's so fast, it can't get out of its own way! The Valid simulation software would run quite fast, except that its disk is limited. UNIX is reinforced with a windows interface. I find it judicious to use five windows (although one just tells me about lunch time). The big one runs Valid's Graphics Editor which does schematic capture, although I'm using it more as a design tool.