Re: CMFORTH

From - Mon Jul 28 12:06:54 1997
Path: uunet!in3.uu.net!204.94.214.20!news.sgi.com!howland.erols.net!infeed1.internetmci.com!newsfeed.internetmci.com!news1.best.com!nntp2.ba.bes
From: "Marc de Groot - Replace username with my first name for all e-mail -"
Newsgroups: comp.lang.forth
Subject: Re: CMFORTH
Date: Mon, 28 Jul 1997 08:22:52 -0700
Organization: Immersive Systems, Inc.
Lines: 151
Distribution: world
Message-ID: <5ridf2$6mb$2@nntp2.ba.best.com>
NNTP-Posting-Host: isi.vip.best.com
X-Newsreader: Microsoft Outlook Express 4.71.1008.3
X-MimeOle: Produced By Microsoft MimeOLE Engine V4.71.1008.3
Xref: uunet comp.lang.forth:35173

>   Around five years ago I thought cmForth was so old-fashioned
>that it would be replaced with something better. But I still see
>favorable references about it. So now I can't decide. Should
>cmForth be studied and documented to make a new and better Forth for
>current computers, or should it just die a natural death from old age
>and lack of interest?

 My opinion is that the weekend or so that one would take to
learn cmForth would be time very well invested.

I first encountered cmForth and the Novix chip in the early
eighties. I bought one of the evaluation kits with cmForth in
ROM, and (amongst other things) wrote a Forth-83 standard
interpreter for the system which became a commercial product.

I regret that the metacompiler in my Forth system, which was
built directly on top of cmForth, used older, better known, and
far less elegant techniques than Chuck's code. :-) By the time
I understood cmForth's metacompilation (which is far simpler
than one's intuition would suggest) I had invested a fair
amount of time and energy in writing my own metacompiler.

Chuck has a unique way of looking at and resolving software
engineering problems. One thing about Chuck that's unusual is
his patience with his own thought processes--patience is not a
prevalent trait among computer people! Chuck seems to derive
great satisfaction from contemplating relatively small problems
for periods as long as years, and he feels like he has really
done something interesting when he has shortened the program by
40% and has somehow managed to add some functionality that
makes the software more useful.

Any programmer with some passion for the craft and a supply of
caffeine can implement bells and whistles at record speed,
writing code that gets the job done--and gets it done
today--but few of us have approached the design (and
redesign) of software the way Chuck does.

Mike Perry relates a story about Chuck's CAD system which ran
on the board whose design it had been written for. Chuck
showed Mike that he had written the core of the CAD application
in about five lines of code. When Mike asked him how long it
had taken, he replied, "Oh, about two years."

cmForth's metacompiler is the result of one of Chuck's long
contemplations. It reduces the largest headaches in
metacompilation to fewer than twenty lines of code, all told.
To make that possible, the structure of cmForth departs
somewhat from the familiar.

The fundamental issue in metacompilation is this: Forth
systems, in most cases, are not designed so that compilation
outside the running system's dictionary is as straightforward
as compiling into its dictionary. The biggest reason for this
is that the programmer now has two dictionaries that must be
considered; the host (or running system's) dictionary, and the
target (or metacompiled system's) dictionary.

Think about what happens when a colon definition is compiled
into the target system. The next symbol in the input stream
has to be parsed and compared against the name fields in the
dictionary so we can find the address to compile. Which
dictionary gets searched? Why, the target dictionary, of
course; we need the address of the Forth word in the target,
not the one in the host, if we want the metacompiled system
to run once the host has terminated! Okay, so we search the
target dictionary for any name we encounter in the input
stream? Well, not quite. What if the word is IMMEDIATE, that
is, what if the word executes at compilation time? In that
case, we can't look up the address of the word in the target
and pass that address to EXECUTE. For all we know, we are
cross-compiling for another processor, and the copy of BEGIN or
UNTIL that's in the target dictionary will crash the system if
we try to execute it.

The above example describes the behavior during
compilation--what about during execution? Well, if we're
looking up words to execute them, they had better be in the
host system, or we had better have a strong reason for
believing that code in the target can be safely executed during
the meta- compilation. Okay, so outside of colon definitions,
we always look up words in the host, right? Well, no. It

depends. Let's say we're using "tick" to get the execution address of a word. Why are we doing that? One possibility is we want to get the execution address of a word and store it in a variable, so it can be fetched and passed to EXECUTE at some future time. If we're patching a variable in the target system, "tick" needs to search the target system for the address of the word in the target. If we're trying to find information about the current state of the running Forth environment, we will likely want to look in the host, where the active Forth system's data is stored.

These considerations often lead the metacompiler designer to write separate versions of words such as , C, ALLOT CREATE "tick" -- and all of the words that depend on these definitions. In some systems, the new set of words is given a distinct set of names indicating it acts on the target system...names such as ,-T C,-T ALLOT-T etc. are common. In other systems, the designer keeps the names the same as the host counterparts, but puts the new set of words in its own vocabulary, and makes use of the vocabulary search order to enforce proper dictionary searching.

Chuck's careful approach to the design of cmForth, his patient contemplation of the issues in metacompilation, and an unwillingness to "settle for less" led to a design and implementation that are cleaner and more elegant than any other that I have seen.

cmForth is roughly ten pages long, written the old-fashioned way, three 16-line disk "screens" to a page. Chuck was proud of the fact that he could recompile cmForth with itself, reading the source off of a 1.44-MB floppy, and the system would completely recompile and prepare a memory image of the new Forth system *before the floppy had spun all the way up to speed*. There's, like, nothin' there, man!

I believe the biggest hurdle to understanding cmForth is Chuck's very terse (read: undocumented) coding style. Certainly, Ting has made efforts to present the material in a somewhat more palatable form. I remember hearing that Jay Melvin, one of Chuck's neighbors (now since passed away?) had annotated and expanded the source code for cmForth. I wonder if anyone has a copy of that... I'm assuming that the legal status of cmForth hasn't changed, and that it is still freely available at no charge?

Thanks much to Frank Sergeant for Pygmy, which kept me entertained for a while this past winter. I've been using Pygmy to build a tiny DOS TSR program. I have been using it to analyze network traffic for multi-user video games. I need something that is very flexible (so a command prompt at run-time is just the ticket!) and the memory footprint has to be as small as possible, since these 32-bit DOS video games tend to be big consumers of low memory on a PC. Aside from its small size and decent features, it *screams*! A very efficient little execution engine, with direct threading. It might be instructive to implement a few of Java's primitives as code words in Pygmy, and compare the execution times...

-Marc de Groot

turingtest@immersive.com
^^^^^^^^^^ Replace this with my first name or mail won't get to me.