Demise of the Metacompiler in cmForth

Jay Melvin, RPCV

Abstruct

Metacompilation is a technique of regenerationg 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.

メタコンパイルとは、コンピュータの実行プログラムを自分自身でコンパイルして新しいシステムを再生成する技術である。 従来のメタコンパイラは、メモリ空間演算子を操作してネイティブメモリ、ホストメモリ、ターゲットメモリの違いを識別し、各空間に同じソースコードのほとんどを使用できるようにする。本論部ではメタコンパイラの機能を説明する。Forthの革新と簡略化は、メタコンパイラのメモリマネージャの新しい実装にかかっていることが示されています。用語集も提供します。

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.

メタコンパイラ(1)は、人間が書いたForthシステムのソース記述を、機械実行可能なオブジェクトコードに変換するプログラムである。Forthシステムのソースを再コンパイルできるため、修正や拡張が容易であり、あらゆるレベルのカスタマイズが可能である。

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.

メタコンパイルは、アプリケーションソリューションの設計過程で日常的に発生するいくつかの問題を解決するために使用されます。通常、プログラミングツールは汎用なものであり、特定の目的のために研鑽を積む必要がある場合があります。プログラムのカスタマイズは、ディスクドライブのコントローラ、ROMやRAMの調整、異なるプロセッサ、バグの修復、オペレーティングシステムや言語などの新しい機能の追加など、環境の変化から生じます。アプリケーションにおけるこのような顕著な用途の違いは、メタコンパイラで新しいプログラムを構築することで容易に対応することができます。なお、エグゼクティブ(核)にコンパイルされるソースコードは、ネイティブ、ホスト、ターゲットの各プログラムでほぼ同じであり、違いがあるとすれば、先ほど述べたシステム的な変更に起因するものです。

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.

ネイティブForthは、アプリケーションの設計、開発、デバッグに使用されるシステムです。アプリケーションソリューションに別のForth(新しい開発環境)が必要な場合、メタコンパイラがロードされ、ホストのメモリにターゲットイメージを構築するために検索やその他の方法で使用できるホスト辞書項目を提供します。ホストのメモリとは、メタコンパイラが占有するForthのネイティブメモリである。最終的に、ホストのターゲットイメージは、それが実行されるターゲットコンピュータにインストールされます。ターゲットコンピュータは、ネイティブが使用するのと同じハードウェアかもしれませんが、ターゲットランタイムでは、この新しいシステムが配置されます。

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

従来のForthプログラム（2）では、メタコンパイルのメモリ管理は、以下のようになります：

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.

A. コンマ、フェッチ、ストア、ムーブ、ダンプなどのメモリ演算子、各種コンパイラ、アセンブラ、フロー制御指令が、ネイティブ、ホスト、ターゲットメモリ空間で動作するように個別に定義されている。 B. ネイティブ空間、ホスト空間、ターゲット空間で動作する、適切なバージョンのCREATE：

1. フィードバックするための対話的解釈のための名前フィールドをコンパイルする。
2. ROM/RAM分離、対話的解釈、インクリメンタルコンパイルのために、各単語を前の単語とリンクさせる。
3. 編集のための "locate "ポインタをコンパイルし、対話的にコンパイルする。
4. 各単語の種類（変数と定数や配列の違いなど）に応じた適切なランタイムアクションをコンパイルする。
5. 未完了のプログラムに対するターゲットアドレスの算出。

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.

実行中のNative Forthがあれば、メタコンパイラをロードして、これらのホストメモリ管理サービスを提供するようにする。コンパイラのエクステンダである`DOES>`の時間的動作とそのコンパイルについては、本稿では触れません。ターゲットの核がメモリに配置され、次にメタコンパイラのコンパイラがターゲット自身のコンパイラとターゲットのアプリケーションを配置する。異なる時間に実行されるプログラムは、メモリの異なる領域に保持される。従来のForthsでは、ターゲット語の敷設場所を制御するメモリマネージャはT、「to target」で、ホスト側の辞書からのターゲット辞書のオフセットを計算する。メタコンパイラでは、演算子や`CREATE`はすべて`T`を使って再定義されます。相当な量のソースコードが必要で、それを理解するにはそれなりの勉強が必要です。

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 amFORTH, the entire metacompiler is reduced to the following:

Forthエンジンの発明は、仮想マシンの再考を必要としました。後知恵で、前述のメモリ管理の機能はすべて別のレベルで実行でき、複雑さを大幅に軽減できるというChuck Mooreの洞察（3）を共有することができます！ cmFORTHは、メタコンパイラプログラムを1語`{`、「スイッチ」に置き換えています。amFORTHでは、メタコンパイラ全体が以下のように削減されます：

```
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", ie., 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, 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.

"スイッチ"により、ネイティブコンパイラは、可能な限り元のソースコードを再利用するという美徳を維持しながら、ホストとターゲットのコードを生成することができます。倍長変数H'はターゲット辞書を指し、再配置オフセットを含む。dAは「デルタアドレス」を含む変数で、つまりホスト辞書ポインタとターゲット辞書ポインタHとH'の差である。2つの辞書ポインタは、`{`または`}`が呼び出されるたびに切り替わります。これらは制御空間に切り替わり、新しいメモリ演算子やホストまたはターゲットのための別の`CREATE`なしで、ネイティブバージョンで通常通りコンパイルが行われます。ターゲットオブジェクトの再配置は、DUMPのようなツールの使用を簡略化するために選択されます。ターゲットイメージの新しいバージョン、例えばDUPをネイティブのDUMPで調べるには、ネイティブバージョンのアドレスにオフセットを追加するだけでよい。プログラムの違いは、アドレスの桁数で一目瞭然です。

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!

スイッチは、コンパイラやツールを、ネイティブ、ホスト、ターゲットの3つのプログラムに関連する適切なメモリ空間にリダイレクトするだけで、DOES>を除くすべてのメタコンパイルのメモリ管理機能を実行することができます。このように、メタコンパイラはForthのどケチ精神によって縮小され、1行のソースで表現されます！

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., FPC are familiar examples. (3) cmFORTH was the first (1986) programming environment for the Forth Engine.

(1) FORTH, Inc.のDean Sanderson(1977-8)が設計したターゲットコンパイラなど。"Target compiler "はFORTH社の商標である。 (2) FORTH社のpolyFORTH、Laxen & Perry社のF83、ZimmerらのFPCなどが有名です。 (3) cmFORTHは、Forth Engineの最初の（1986年）プログラミング環境である。

Glossary

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

`WORD` 機械命令の名前付きコレクションで、名前は機能を示唆する。

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

`コンパイラ(COMPILER)`は、機械命令を将来使用するためにメモリに格納するワードである。

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

`PROGRAM` ユーザに何らかのサービスを提供する単語。例えば、開発環境、オペレーティングシステム、言語などはプログラムである。

`NATIVE` computer hardware and software used for application development.

`NATIVE` アプリケーション開発に使用されるコンピュータのハードウェアとソフトウェア。

`TARGET` computer hardware and software under construction for future use.

`TARGET` 将来の使用のために構築中のコンピュータのハードウェアとソフトウェア。

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

`HOST` TARGETを構築するために使用されるコンピュータのハードウェアとソフトウェアで、ネイティブのPROGRAMとメタコンパイラの両方が動作する。

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

`NUCLEUS` リアルタイムエグゼクティブ、スタックマニピュレーター、算術演算子、論理演算子。

`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.

`INTERPRETER` ワードまたはスタック、数を実行するプログラム。入力はキーボード、ディスク、または他のI/Oポートからプログラムの制御下に置かれる。

`DICTIONARY` program memory organized as linked WORDs.

`DICTIONARY` リンクされたWORDとして編成されたプログラムメモリ。

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

`CREATE` すべてのWORDをコンパイルするために必要なキーワードで、VOCABULARY内の前のWORDにリンクしている。

`LINKER` every incremental compilation provides connection of the new routine to extant WORDS in the dictionary, one of the functions of CREATE.

`LINKER`は、インクリメンタルコンパイルのたびに、新しいルーチンを辞書の現存するWORDに接続するもので、CREATEの機能の1つである。

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

`ランタイム`(RUN TIME): 単語がアプリケーションを実行する時間「明日」。

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

ネイティブシステムでワードが構築される時「現在」、ネイティブ自体が構築された「昨日」とは異なる時。

`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.

`RUN`または`EXECUTE`は、インタプリタがあるメモリ空間を指し示し、そこで指定した動作の実行を強制するワードを見つけることを意味する。

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

`LOAD` or `COMPILE` or `lay down into memory` は、エディタでのプログラミングとは異なり、インタプリタが行うプログラミング動作を指す。

> 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.

> ジェイ・メルビンはTracor Ultron LabsのためにRTXプロセッサ上で動作するコードを書いています。彼はまた、MAXTOR Corp.とFORTH,Inc.でForthを使って仕事をしていました。彼は、Computer CowboyのcmFORTHの最初のリリースのためのシャドードキュメントを書きました。彼は、平和部隊の志願兵退役者です。

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) FORTHought and UNIX. An opportunity to get an update on the state-of-the-art.

最近、私は本当に奇妙なことをしています。他の人のソフトを使っているんです。Forthの前の時代に戻ったようだ。あのね、私は別のチップを設計したいんだ。これはルールに従うということだ。今のところ、それはワークステーション、Pascalプログラム（だと思う）、FORTHought、UNIXということだ。最先端の情報を入手するチャンスです。

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.

ボブ・マーフィーと私がNovixチップを作ったとき、ボブは彼がよく知っているハイランドシステムを動かした。私は遅くて不器用なインタラクションをハックすることができず、Forthでのシミュレーションに集中した。今回、私はホットシートに座り、システムに対処する知恵と忍耐力を身につけました。20年もForthと付き合ってきたのに、どこまで疎くなってしまったのか、興味と不安でいっぱいだった。何も変わっていないことを知った時の安堵感を想像してください。

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.

昔はメインフレームを使っていたが、今はワークステーションを使う。ディスプレイは珍しいものでしたが、今ではカラーになっています。オペレーティングシステムはOSで、今はUNIXです。言語はFortranからPascal(C?)へ。しかし、何も変わっていない。1969年の時点で1989年をよく予測できたと思う。結局のところ、問題がないところには変化の必要はないのだ。

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.

私はSun-4ワークステーション、SPARCマイクロに8MバイトのRAMと370Mバイトのディスクをイーサネットでバックアップして使っています。これはきっと現在のハードウェアだ。あまりに速いので、どうにもならない！ Validのシミュレーションソフトは、ディスクに制限があることを除けば、かなり高速に動作していると思う。 UNIXはウィンドウ・インターフェースで強化されています。 私は5つのウィンドウを使うのが賢明だと思う(そのうち1つはランチタイムを知らせてくれるだけのものだが)。 大きなウィンドウはValidのGraphics Editorで、回路図のキャプチャができるようになっているが、私は設計ツール以上に使用している。