Conference Abstracts

The following abstracts are from presentations made at the 1985 FORML Conference, November 29-December 1, 1985, Asilomar Conference Center, Pacific Grove, California.

Yet Another Recursive Decompiler*

Richard Astle

Interpretive Logic

Wil Baden
Costa Mesa, CA

Interpretive logic for conditional execution, conditional compilation, and text editing extend Forth to be responsive with modern system requirements.

Design of the CICFIG Forth Target Compiler

Sidney A. Bowhill Howard H. Robinson Philip D. Morse III
University of Illinois
Urbana, IL

During the past year, the Central Illinois Chapter of the FORTH Interest Group has developed a target compiler, which has just appeared in print (Robinson et al., 1985). The need for it arises from the design of portable data-acquisition equipment where compactness of code is essential. The work was defined and monitored by the three authors of the published paper at about six monthly meetings, attended intermittently by about five other members of the Chapter. The actual code was written by Robinson and Morse.

The purpose of the present paper is to discuss the design philosophy of the compiler, to describe some of the difficulties that were encountered, and to invite comments and criticisms.

Self-Understanding Programs

Mitch Bradley

This presentation describes a wordset that makes it possible to write portable programs which understand themselves. The most obvious example is a Forth decompiler, which is traditionally non-portable.

A Keyboard Monitor for FORTH

Lance Collins
Micro Models Pty Ltd
Glen Iris, Victoria, Australia

Vectoring KEY enables the implementation of a simple but powerful monitor over keystrokes entered at the console which is useful in the kernel and readily available to application code.

LOGO in FORTH - A Role for Stack Frames and Local Variables

Lance Collins
Micro Models Pty Ltd Glen Iris, Victoria, Australia

Implementing a LOGO compiler in FORTH for the Microbee computer required the use of stack frames and local variables. There are some useful ideas in this work relevant to the use of local variables in FORTH, particularly where the word using local variables may be used recursively.

*No abstract available.

Micro Models FORTH: a Public Domain FORTH Without Screens

Lance Collins
Micro Models Pty Ltd
Glen Iris, Victoria, Australia

Micro Models FORTH is a Forth without screens, relying on operating system files for source and data storage. A cross-compiler supports easy modification of the system and porting to other environments. The system sources are heavily commented and logically arranged for ease of understanding. Except for screens the system follows the Forth-83 standard.

A Simple Interrupt System for Forth

Lance Collins
Micro Models Pty Ltd
Glen Iris, Victoria, Australia

What to do when the ability to interrupt a running process is required and no hardware interrupts are available?

One way is to recode `NEXT` to check the keyboard for input. On a CP/M system this adds a very large overhead to the `NEXT` loop. The practical solution to avoiding this overhead was to have the special `NEXT` code but only use it from one commonly used word. `DROP` was used, and so far we have not found any significant application code which does not use `DROP` frequently enough for our purposes.

Improvements in Error Handling

Loring Craymer
Division of Biology
California Institute of Technology
Pasadena, CA

Error trapping mechanisms can significantly improve the user interface of a Forth system. Many vexing system crashes can be avoided by setting the appropriate error traps. I have adapted an error trapping mechanism to a structured syntax which is convenient for occasional use. I have coupled message formatting tools to the error trapping mechanism; the message formatting tools are useful for reporting multiple arrors and for constructing iterated error messages.

For multitasking, the message formatting tools make it a simple matter to delay the reporting of errors in background tasks until they can be reported without risk of garbling console output. I have also developed a multitasking version of `QUIT` which allows background tasks to abort gracefully whan an error is encountered rather than crashing the system. This version of `QUIT` also obviates the explicit use of `STOP` when a task terminates successfully.

A Portability Wordset

Loring Craymer
California Institute of Technology
Pasadena, CA

Now that 32-bit Forth implementations are common, it seems appropriate to discuss portability concerns. I have compiled a small wordset which deals with the problems of differential cell and "token" sizes, alignment of addresses on the return stack, and data storage in `BLOCK`s. Comments and suggestions for additional words are welcome.

Word Search, Filter, and Structure Tools

Regan Fuller
Hewlett-Packard Co.
Palo Alto, CA

When Forth vocabularies become large, it is difficult to remember so many word names and what they do, even if you are fortunate enough to have exhaustive documentation. This paper will discuss a group of utility words that make searching through a large vocabulary much easier, as well as some for looking at word structure and size. It will also discuss some vocabulary search filters to find different word types. There is also a string finder. These tools have been implemented on 32-bit Creative Solution, Inc. Multi-Forth Version 2.07 running on a HP9836 computer.

Forth Primitive Pairs Profiled

Rod Goodman
Electrical Engineering, CALTECH
Pasadena, CA

The technique of profiling a Forth application is of particular interest to designers of Forth processors. The architecture of a Forth machine is influenced by the software primitives that need to be implemented, and vice-versa. Choosing a suitable set of primitives can only be sensibly done if the frequency of Forth primitives is known. Profiling techniques can be used to establish these frequencies. Once the architecture is determined, it is possible to postulate that sequences of two or more primitives can be combined to form a new primitive. The objective here is to eliminate the memory accesses involved in fetching the primitives individually, and thus make the machine faster. However, this is only worth doing if the frequency of occurrence of the proposed primitive group is high enough. Having a group combined into one primitive, just because it fits the existing architecture, is no good if that group occurs with vanishingly low probability! In this paper we examine the frequency of occurrence of pairs of Forth words via the profiling technique. Results of profiling different applications are shown, and a "top-ten" of useful primitive pairs is formulated.

Extending FORTH Control Structures into the Language Requirements of the 1990's

David W. Harralson
Mephistopheles System Design
Yorba Linda, CA

The FORTH conditional and looping structures have been extended to include a generalized control structure while retaining existing FORTH words. Simple changes to existing FORTH programs adapt them to the new structure.

Forth Coding Conventions

Kim R. Harris
Palo Alto, CA

This paper contains proposals for several conventions for Forth programming. The intention of this paper is to present guidelines, not rigid constraints.

The purpose of these conventions is to lead to a common style for Forth programs. Conventions increase the bandwidth of communication between programmers. Coding conventions concern only the physical appearance of programs and affect neither the programming techniques used nor the creativity brought to solve problems. Commonly used and accepted conventions allow one programmer to read another's program quickly and accurately. They also promote the publication of programs and job interchangeability for programmers.

Using the STRUCTURE-TOOL Program to Automatically Construct Structure Charts of Forth Programs

Kim R. Harris
Hewlett-Packard Co.
Palo Alto, CA

This article describes how to use a software tool named STRUCTURE-TOOL. It is used to analyze a Forth program's procedures and report on its structure. It can produce several reports: a structure chart, a fan report, and a data access report. Each report is explained with examples.

Options are described which allow the tool to analyze subsections of a program, exclude classes of words, and limit the number of levels reported. Suggestions are made for selecting options.

Analyzing Large Forth Programs by Using the STRUCTURE-TOOL Program

Kim R. Harris
Hewlett-Packard Co.
Palo Alto, CA

This article describes a method for analyzing programs written in the Forth computer language. It provides a "road map" to help you learn an unfamiliar program's instructions. The STRUCTURE-TOOL program is used to automatically determine the structure of a program's procedures and named operands and to produce several reports including a structure chart.

The method primarily uses only a program's structure to find its major functional divisions. The method is illustrated by applying it to the STRUCTURE-TOOL program itself.

Internal Documentation for the STRUCTURE-TOOL Program

Kim R. Harris
Hewlett-Packard Co.
Palo Alto, CA

This article explains the algorithms and data used in the STRUCTURE-TOOL program and also includes information about: verification testing, reconfiguring, and transporting it to other Forth systems and language dialects. It explains how to configure the program for different printers and how to change the maximum number of words that can be analyzed. The implementation is based on the Forth 83-STANDARD; non-standard words and implementation-dependent details are presented.

The "workers-list" data structure is the main repository for information about the words being analyzed. Its arrays and operations are described. Three passes over the words are explained: the learning pass, the studying pass, and the reporting pass.

Source Listing for the STRUCTURE-TOOL Program

Kim R. Harris
Hewlett-Packard Co.
Palo Alto, CA

This article contains a source listing for the STRUCTURE-TOOL program version 2.0. It is implemented on the F83 Forth system version 2.10, and it is configured for an Epson MX-80 printer. The language dialect is "Forth 83-STANDARD with non-standard extensions".

Separate Code and Data Spaces in Forth

Jesse W. Hartley
Dalton Research Center
University of Missouri-Columbia
Columbia, MO

Partitioning memory into separate instruction and data address spaces is desirable in many situations and is supported by many current hardware architectures. This separation is routinely done in read-only memory based, specific applications, but also offers advantages in general-purpose Forth systems. Among these advantages are the ability to write-protect code in multiuser systems, and increasing the address space available for large data structures. The primary areas of impact are those concerned with Dictionary construction, i.e. the interpreter, variables, vocabularies, and "state smart" words. Modifications include a high level interpreter using a string stack for text handling instead of the PAD buffer, and a "user variable" pointer system for named variables.

A Forth Component Library for Off-the-Shelf Distribution of Modules

John S. James
Santa Cruz, CA

A system now being developed provides comprehensive support for moving large components of software systems among different institutions and work groups, across different Forth dialects and implementations, and across different operating systems. The goal is to create a marketplace for off-the-shelf modules which can be used in many different environments as black boxes, with no need to know or change any of their internals.

Threaded Binary Trees in FORTH

Robert E. Klebba
Nicolet Analytical Instruments
Madison, WI

A Forth implementation of threaded binary trees and orthogonal binary trees will be presented. Binary trees can be used to structure data in data bases, generate and maintain symbol tables during compilation or as in the application presented, simply sort lists of numbers. A relatively simple extension to binary trees is the orthogonal binary tree where different trees structure the same sets of data but on different keys. The advantage of the use of binary trees in sorting is in its execution time. It can be shown that for the average binary tree sort, the execution time is proportional to $n \log_2 n$ (where n equals the number of data sets), whereas that for a bubble sort is proportional to n^2. Although threaded binary trees require more memory space per node than for an "unthreaded" binary tree, the threaded binary tree structure minimizes parameter stack and return stack usage.

BNF: A Parser Written in Forth

Leonard Morgenstern

BNF, named in honor of Backus and Naur, is a parser, written in Forth, capable of analyzing complex input. Parser words can be intermixed with ordinary Forth words, so that BNF can analyze the text for correctness and also act on it. BNF provides for parallel, series, repetition, and nulls, the latter being elements that must or may be absent. BNF does not conform to the conventional requirement of "one symbol look-ahead without backtracking", a feature that provides certain advantages.

An Approach to Natural Language Parsing

Jack Park
Brownsville, CA

A description of the design of an expectation-based parser is given. The parser uses a three-part structure consisting of primitive structures, expectation procedures and heuristics, and a lexical dictionary. By use of this structure in a Forth environment, a self-parsing vocabulary is built.

Performance Analysis in Threaded Code Systems

Michael Perry
Berkeley, CA

The importance of performance analysis to the iterative design process is discussed. Several techniques for performance analysis in Forth systems are described. Some related debugging techniques are mentioned.

Knowledge Representation in Forth: "What is a *fact*? Well, it depends on the *time*..."

Dana Redington
Sleep Research Center
Stanford University School of Medicine
Stanford, CA

The representation of knowledge in computer software can be accomplished by means of a variety of programming languages beyond Lisp and not exclusive to Forth. While Lisp has been the pen-ultimate language for Artificial Intelligence, other languages have been used to develop expert systems, such as Lisp's derivatives Logo and Prolog, and non-list oriented languages, such as Fortran, Basic, Pascal, and C. However, Forth provides the unique opportunity to transform a totally extensible environment into a knowledgeable system that can manipulate facts as objects with a scant amount of high-level code and yet retain the integrity (freedom) of Forth. This paper presents a brief discussion of knowledge representation in Forth and illustates a one block extension to entertain Facts which form a partial basis for machine intelligence.

A Prolog Interpreter

C. H. Ting
San Mateo, CA

Prolog is an interesting language with simple syntax structure. It is rather straightforward to implement it in Forth. Since most of the work in interpreting Prolog queries is in string comparison, the interpreter can be constructed by running two pointers, one for the query string and the other for the data base. This method thus eliminates the need to build data structures according to the data types used in Prolog. In this paper, only relations are allowed in the data base. Both the "does" and "which" queries are implemented to use the data base. Work to implement rules in data base is in progress.

A Forth LISP

Martin J. Tracy
Micromotion
Los Angeles, CA

Forth is an extensible language and can be extended to handle lists of facts. The LISP language has many good list operators which can be translated to Forth in a straightforward manner. By choosing appropriate data structures and algorithms, Forth can have all of the power of LISP without the performance penalties. On the other hand, Forth pays for this with greater complexity and the accompanying risk of producing incorrect code.

This paper, which freely combines source code with text, describes a hybrid Forth/LISP list handler. It discusses list data structures, list primitives, "aliases", dynamic garbage collection, determining symbol uniqueness, and other implementation details. As a demonstration of the power of the hybrid, it is itself used to implement the "micro-LISP" described in Winston and Horn's *LISP*, 2nd edition, (Addison-Wesley, 1984).

The POSTMAN: A Multiprocessor, Multiprocess Communications Environment

Jon N. Waterman
FORTH, Inc., Hermosa Beach, CA
Howard C. Goodell, John R. Langley
PT Analytical, Wilminton, MA

A 375 kilobaud serial link connects a network of 8031 control processors to an 8031 link controller in automated instruments. The link controller communicates with a 68000 host via a bank of shared memory; the protocol for controlling transfers through this shared memory is decribed. Both the host and many of the 8031 nodes are multi-tasked; the protocol provides for transfers between any task on the host and any task on any node, and any number of such conversations may proceed simultaneously. The goals are achieved via a collection of control parameters for each communications channel, called a mailbox. Two copies of each mailbox exist; one in the host's regular memory which host tasks access, and one in shared memory which the link controller accesses. A host routine called POSTMAN scans the mailboxes rapidly whenever a transfer is requested in either direction and copies both control information and data; the link controller, called the COURIER, communicates with the requested node and task to complete the transfer.

Open Metacompilation

Martin Worrell
Metaforth Computer Systems Ltd
Hull, England

Metacompilation is the process of re-compiling a complete self-contained Forth system and application, to be run on the host processor.

Conventional metacompilers either use a specialised Forth kernel source compiled in one pass, or use a normal "clean" Forth kernel source compiled using a multi-pass compiler. Both of these methonds produce a system which can only be run and tested once it has been compiled, relocated, and reloaded in its entirety. These "blind" compilation techniques are difficult to verify due to the amount of work that is done before any of it can be tested.

The technique described in this paper compiles a "clean" Forth kernel source and application and then modifies the compiled code to produce the final system.

The important feature of this proposal is that at all stages the system may be used, examined and tested in the normal Forth way. The only "blind" step is the final relocation of the completed system.

The following abstracts are from presentations made at the 1985 euroFORML Conference, October 25-27, 1985, Stettenfels Castle, West Germany. The papers from this conference are included in the Proceedings of the 1985 FORML Conferences.

English as a Second Language for Forth Programmers*

Wil Baden
Costa Mesa, CA

Formal Rules for Phasing*

Wil Baden
Costa Mesa, CA
Ken Clark
Garden Grove, CA

Data Collection in Elementary Particle Physics with 32-bit VAX/68K Forth

Ralph Haglund
CERN
Geneva, Switzerland

Two 32-bit Forth-83 systems have been developed, running identical Forth code, with identical development environment.

ISD - the ideal complement to XTC

A.P. Haley, H.P. Oakford, C.L. Stephens
Computer Solutions Ltd
Chertsey, Surrey, England

The availability of a high level programming language such as polyFORTH on a wide range of microprocessors coupled with the low cost of the current generation of PCs suggests that significant savings can be made if Cross Target Compilers (XTCs) can be made available on the PC. However, the use of a Cross Target Compiler immediately loses the interactive development of hardware oriented parts of an application that is so characteristic of Forth and from which so many of its gains derive.

This paper will decribe a package called In Situ Development (ISD) which aims to provide an easily implemented technique to allow developers of standalone applications hardware and software to take advantage of Cross Target Compilers without losing direct contact with their hardware.

Forth and Artificial Intelligence *

Robert La Quey

A Forth Driven, Networked System for Applied Automation

Donald C. Long, Jr.
Florida Drum Co.
Pine Bluff, AR

Finding flexible and cost effective ways of interfacing microcomputers for data acquisition and control has been a stumbling block for many applications.

The availability of low cost Single Board Computers and intelligent input/output systems provides exciting new capabilities for the automation of systems and entire facilities.

This paper describes an applications command language known as the master control program. Supported hardware presently includes the Optomux family of intelligent interface boards. The advantages, system performance, and limitations of this form of networked distributed control are reviewed.

Generic Operators *

Terry Rayburn

Control Simulation for Tape Deck

Ludwig Richter-Abraham

The following screens show a simple way to control a stereo-tapedeck or something like this. In spring '84 I fixed an auto-repeat-control into an electronic tapedeck and wrote the program in assembler because I had no Target-compiler in FORTH. But this year I got one and to check out the differences of the two approaches it was restyled in FORTH.

Preliminary Report on the NOVIX 4000

C.L. Stephens and W.P. Watson
Computer Solutions Ltd
Chertsey, Surrey, England

The NOVIX 4000 is a true FORTH processor and is capable of in excess of 10 million FORTH instructions per second. It is implemented as a gate array.

This paper will introduce the architecture of the chip, its hardware configuration and the software support facility provided with it. Details of a number of benchmark programs will be given and these will be used to identify particular application areas for the chip.

A Set of Forth Words Lets You Do Network Analysis

Jens Storjohann
Deutches Elektronen-Synchrotron DESY
Hamburg, W. Germany

Most basic tasks in electrical network analysis are calculations of driving point impedances of one-ports and the transfer functions of two-ports.

The program NAOMI (Netzwerkanalyse ohne Matrizeninversion / network analysis without matrix inversion) uses a simple language to describe components and networks and to immediately invoke suitable arithmetic operations.

The following example calculates the admittance of a one-port built by connecting two series RC-circuits in parallel.

```
60 HZ IS-FREQUENCY R1 C1 ++ R2 C2 ++ ||
```

This approach avoids in contrast to systems like SPICE numbering of nodes and storing and inverting large matrices.

Forth Language Extension for Contolling Interactive Jobs on Other Machines.

David K. Walker
Virtual Walker & Walker
Tertnes, Norway

A Forth application on an IBM PC/XT is described for 1) collecting, editing and generating input for a large model of the Nowegian economy used by the Norwegian government, 2) transferring this information to a mainframe, and 3) running interactive jobs which check the input, process it further and send it on to another mainframe where the economic model equations are solved and result tables are written. The application emulates a person operating a computer terminal. Forth techniques illustrated include a finite state decription language extension for controlling general interactive jobs on other machines. Anticipated further developments of this concept include controlling and logging the entire model solution process on several machines.

RTDF: A Real-Time Forth System Including Multi-tasking

H.E.R. Wijnands and P.M. Bruijn
Control Engineering Laboratory
Delft University of Technology
Delft, the Netherlands

This paper outlines a real-time Forth system, named RTDF. RTDF is intended for use as a development tool for single processor control systems. Due to the general language concepts applied, it has also been proved useful for discrete system simulation and other concurrent programming needs.

RTDF offers multiple task declaration, initiation and priority assignment. Synchronization and communication between tasks can be performed by the use of semaphores and monitors. Real-time facilities include the general use of timers and delay statements. An application running on a Z80 at 4 MHz may utilize a 5ms sample time, maintaining an efficiency better than 95% for any number of tasks in any state.

Event Driven Multitasking - A Syntax

Jens Zander
SPECTRA-Colon Systems, AB
Linkoping, Sweden

In many real-time programming applications, programs are to be initiated at the occurrence of some internal or external event. Examples of such events are the real-time clock reaching some predetermined value, an I/O device needing service or some logical condition being fulfilled. The classical way of solving some of these problems is to use hardware interrupt signals. In this paper, situations are investigated where for various reasons interrupts cannot be used. An example is when the condition tested is a very complex one. A FORTH-syntax for general event handling is proposed, including the structures `EVERY`, `AFTER`, and `WHENEVER .... PERFORM`. An implementation for (time-shared) multitasking FORTH systems is sketched.

The Proceedings of the 1985 FORML Conference is available from the publisher, the Forth Interest Group. The price of the Proceedings is $30 domestic, $33 foreign surface, $40 foreign airmail. CA residents add 6% sales tax, 7% in SF. Prepaid orders only in US funds may be sent to: Forth Interest Group, P.O. Box 8231, San Jose, CA 95155 USA.