Towards a Formal FORTH

Peter J. Knaggs
Department of Computing Science,
University of Paisley, High Street,
Paisley, Scotland.
pjk@cs.paisley.ac.uk

Monday 27th September 1993

Abstract

Over the last couple of years several papers have been presented attempting to bring formal methods to FORTH, or applying formal methods to FORTH. This paper brings together many of these ideas and suggests a method by which they could be integrated into a single formally specified and implemented FORTH programming environment.

Much of the theoretical work described in this paper exists in some form. However, the proposed integration of these theoretical ideas has not been investigated any further than this paper.

1 Introduction

Over the last three years or so we have seen several papers attempting to bring the worlds of formal methods and FORTH together. Papers have appeared on the initial investigation into the specification of the FORTH Integrated Development Environment (IDE) [1,2]; to a stack based type theory (suitable for FORTH) [3–9].

Several papers have appeared in the formal methods world that relate to Forth, although the authors may not consider this so. For example, in [10] Spivey describes a simple real-time kernel with a co-operative round-robin scheduler. He goes on to describe the simple prioritised interrupt system provided by the kernel[1]. Indeed this is a good description of Forth's Co-operating multi-tasking system. The scheduler is described in sufficiently abstract terms that most, if not all, current Forth schedulers would meet it requirements.

In [11] Bowen provides a formal description of the Motorola M6800 microprocessor. The intention was to provide a basis for formal proof of software developed for this processor. Bowen says "*more complicated and modern microprocessors such as the 68000 family could be specified in a similar manner. However such processors would require a larger document and more work in order to cover them fully.*" The Forth based processors (such as the F-RISC or Dolphin processors) would make a particularly good processor for such a specification, as it implements the Forth abstract machine directly in silicon.

1.1 Type Checking

There are currently two teams developing formal type checking systems for Forth, one at the University of Tartu and the other at the University of Teesside. Although these teams are working in the same area, and communicate with each other, they are approaching the problem in two very different ways.

Jannus Pöial's paper [3] was the first to show a formal typing system for Forth. Although the work was still incomplete it sparked off an interest at Teesside leading to our paper [6]. After continued work by both teams we have developed two very different typing algebras. The most recent description of which can be found in [9].

[1]It should be noted that a simple abstract model similar to the one presented in [10] could provide a means of standardising the multi-tasking aspect of Forth. This topic was not covered by the ANSI Standard due to its complexity. However, a simple abstract model that does not place any restrictions on the scheduling algorithm, yet provides for the majority of algorithms should be acceptable.

At Teesside we have implemented a version of the type algebra in SML ([8]), and are working on extending Mitch Bradley's SunFORTH to include type checking, without detracting from the FORTH's flexibility. Using type checking we are able to identify programming errors at compile time, rather than having to track down spurious run time errors.

The ANSI Standard's concentration on portability has introduced many new data types (and pointers). This matches well with the idea of checking for the abuse of such types.

1.2 Formal Basis

At Teesside, we investigated the idea of providing a formal basis for the specification of the FORTH IDE. Our initial thoughts where presented in [1] and later in a more complete form in [2].

Even with this formal basis it was not possible to conduct proofs due to the lake of stack typing[2]. It was at this point our investigations were suspended. When Pöial published his work on a type theory suitable for stack based languages [3] he presented us with a method for overcoming the this problem.

Although we have since concentrated our efforts into developing our own type algebra, we now feel that it should be possible to generate a formal basis for FORTH. The idea being that one would take a specification, and refine it into a FORTH implementation. Using the formal base it should be possible to prove that the implementation has the same properties as the original specification, and is therefor a true implementation of that specification.

1.3 Formal Methods

The Formal Methods world has been looking for a way to develop embedded software that can be proven to work. York Software Engineering and the

[2]In would be possible, however the specification would have to be at such a low level of abstraction that it would be too time consuming for sufficiently little result.

Defence Research Agency (DRA)[3] have recently released a system known as SPARK, a method for deriving correct ADA programs from Z specifications.

The Open Systems Federation (OSF) and the DRA are currently attempting to develop an ADA 'producer' for their ANDF intermediate language technology. This compiler is being specified using the RSL formal notation. The idea being that one could use the RAISE method to derive an ADA implementation of a formal specification and prove that it holds the same properties as the original specification.

2 Proposal

In this section we present a project in which the whole FORTH community could take a part. This is a large project and could take a long time to come to fruition. There may be several research projects in the idea presented here.

2.1 Outline

The project is to develop a formal FORTH IDE similar to the OSF/DRA ADA system. This would allow one to specify a problem. And using a known method (such as RAISE) derive a FORTH program that can be proven to be a correct implementation of that specification.

This would start with a formal specification of a stack processor, probably an existing one, or better yet one still in the design stage. Having developed this specification, it should be possible to develop a full ANSI standard FORTH system for it, complete with a formal programming model.

The idea is to provide the formal support for embedded applications. One could develop a formal specification of an application. This specification could then be refined, using the formal model, into a FORTH program. As the

[3]York Software Engineering is an offshoot of the University of York. The Defence Research Agency was previously known as the Royal Signals and Radar Establishment (RSRE) at Malvern.

FORTH system is formally defined we can prove that the resultant program is a true implementation of the original specification.

This project could be achieved in the following way:

1. Develop a formal model of a simple embedded micro-controller.
2. Develop a model for a simple ANSI Standard FORTH IDE. This environment should be formally specified and proven to be complete, consistent and compliant with the ANSI Standard.
3. The FORTH IDE should be implemented, and proven to meet with its specification.

2.2 Argument

Although attempts have been made at specifying hardware before (re Viper), in this project we propose using an existing (or forthcoming) embedded micro-controller. The reasoning behind this, is that the technology already exists, and we are interested in how to best use this technology.

The flexibility of FORTH makes a formal model slightly more difficult than for other languages, however, the underlying abstract machine is particularly well suited for formal specification. With the addition of the type algebra we feel that this is now possible. Such a formal IDE underpinned by a specified micro-controller would make it possible to prove that the full system is a true implementation of the original specification.

This is about as far as we can go before having to specify and prove hardware! The failure of the Viper project [12–14] shows this to be a difficult, if not impossible, task. However given some of the new hardware compilation techniques being developed by the Programming Research Group at the University of Oxford this may become easier in the future [15].

3 Summary

Over the last couple of years papers have appeared on specification of the FORTH IDE; to stack based type-theories. We have looked at some of this work, and some of the work coming from the Formal methods/Safety critical systems people. This was found to be not only applicable to FORTH, but to have a similar intent.

We looked briefly at Spivey's description of a simple FORTH-like real-time kernel complete with a co-operative round-robin scheduler, and at Bowen's specification of the Motorola 6800. This proves that not only is the FORTH abstract machine well suited for formal specification, but it would be of interest to Formal methods/Safety critical practitioners. As there are FORTH engines that implement the abstract machine directly in silicon we could take such a specification all the way down to the processor's instruction set (and possible even further).

We went on to present a project that would bring the different ideas together. The project is outlined by means of a proposal that we (the FORTH community) develop a formal FORTH IDE such that one can use a formal model to derive a FORTH implementation of a specification. It should be possible (using the formal model) to prove that the FORTH implementation holds the same attributes as the original application's specification, and is therefor a true implementation of that specification.

Providing a complete embedded programming environment would make FORTH a more attractive target language. As the IDE has been developed to a standard and has been proven, one need only concentrate on the problem at hand and not on the underlying environment as is currently the case.

References

[1] Bill Stoddart. Specification & Optimisation. In *Proc. EuroFORML Conf.*, September 1988.

[2] Peter J. Knaggs and Bill Stoddart. Formal FORTH. In *Proc. Rochester* FORTH *Conf. on Automated Instruments*, pages 50–55, June 1991.

[3] Jannus Pöial. The algebraic specification of stack effects for FORTH programs. In *Proc. EuroFORML Conf.*, October 1990.

[4] Jannus Pöial. Multiple stack effects of FORTH programs. In *Proc. EuroFORML Conf.*, October 1991.

[5] Peter J. Knaggs and Bill Stoddart. The Cell Type. In *Proc. Rochester* FORTH *Conf. on Automated Instruments*, pages 55–57, June 1991.

[6] Bill Stoddart and Peter J. Knaggs. Type inference in stack based languages. In *Proc. EuroFORML Conf.*, October 1991.

[7] Bill Stoddart and Peter J. Knaggs. The (almost) complete theory of FORTH type inference. In *Proc. EuroFORML Conf.*, September 1992.

[8] Bill Stoddart. Implementation of the FORTH type checker. In *Proc. EuroFORML Conf.*, September 1992.

[9] Bill Stoddart and Peter J. Knaggs. A type signature algebra for stack based machines. *Formal Aspects of Computing*, 5(4):289–98, August 1993.

[10] J. Michael Spivey. Specifying a real-time kernel. *IEEE Software*, pages 21–28, September 1990.

[11] Jonathan P. Bowen. The formal specification of a microprocessor instruction set. Technical Monograph PRG-60, Oxford University Computing Laboratory, 11 Keble Road, Oxford, UK, January 1987.

[12] D. H. Kemp. Specification of Viper1 in Z. RSRE Memorandum no. 4195, Royal Signals and Radar Establishment, Ministry of Defence, Malvern, UK, October 1988.

[13] D. H. Kemp. Specification of Viper2 in Z. RSRE Memorandum no. 4217, Royal Signals and Radar Establishment, Ministry of Defence, Malvern, UK, October 1988.

[14] Donald MacKenzie. Burden of proof goes to trial. *The Time Higher Education Supplement*, March 5 1993.

[15] Brain L. Thompson. Hardware compilation as an alternative computation architecture. MSc. Thesis, School of Computing and Mathematics, University of Teesside, Borough Road, Middlesbrough, Cleveland, TS1 3BA, UK, August 1991.