Semantics of an embedded vector architecture for formal verification of software

A Part III project proposal

G. M. Brown (*gmb60*), Queens' College

Project Supervisor: Dr J. D. Yallop

Abstract

All good implementations of cryptographic algorithms must be correct, side-channel free and fast. Most cryptographic libraries focus on maximising speed by writing hand-tuned assembly. This can introduce subtle bugs that invalidate correctness or introduce side-channels. Whilst tools exist to help formally verify these algorithms, none are designed to target the recent M-profile Vector Extension for the Armv8.1-M architecture. My project seeks to define semantics for these vector instructions, designed to be used for formal verification of software. I will use these semantics to formally verify the correctness of hand-written assembly for cryptographic applications.

1 Introduction, approach and outcomes

Any good implementation of a cryptographic algorithm satisfies three properties: it is correct; it is side-channel free; and it is fast. Hopefully, it is obvious why implementations should be functionally correct. If an implementation has side-channels, then it can leak information about the secrets used, defeating the point of using cryptography. The speed of the implementation is important because executing cryptographic algorithms steals resources from tasks performing "useful work". Ensuring an implementation satisfies these three properties is usually difficult.

Most cryptographic libraries focus on speed, and use hand-tuned assembly code to maximise performance. Truly optimal performance comes when details of the machine microarchitecture are used to reorder instructions to minimise the amount of time the processor is stalled, which can involve reordering instructions until the original algorithm is almost unrecognisable. Additionally, assembly language is notoriously difficult to understand, so these hand-written highly-tuned implementations can have subtle bugs that invalidate correctness or expose side-channels. This is evidenced by the ever-increasing number of CVEs reported for industry-standard software such as OpenSSL and BoringSSL, rooted in their assembly code.

To eradicate these correctness and side-channel bugs with certainty, formal verification is necessary. Formal verification starts with a description of the semantics of machine code – a model of the action of instructions on machine state. This semantic framework is incorporated into a larger logical system, such as a higher-order-logic theorem prover or a dependently-typed programming language. These logic systems can then be used to formally prove that given assembly code satisfies the desired correctness and side-channel-safety properties.

Industry has already created and uses formal verification tools for some architectures. For example, Jasmin is a formal verification tool for x86. It accepts assembly language augmented with logical assertions and high-level constructs like variables and loops. This is converted by Jasmin into pure assembly language output, as well as a proof that all the assertions in the input hold.

An often-overlooked architecture in the field of formal verification is Armv8.1-M. This instruction set is designed for use by microcontrollers, which operate in a resource-constrained environment. In particular the recent M-profile Vector Extension (MVE), which provides SIMD instructions for Armv8.1-M cores, has no known semantics suitable for formal verification.

My project is primarily concerned with the development of semantics for MVE instructions. This will be in the form of operational and axiomatic semantics developed in Agda. These semantics will be used to formally verify the correctness of the implementation of several cryptographic primitives.

2 Workplan

Start	End	Work
2021-12-06	2021-12-19	Familiarise myself with MVE instructions. Begin work on defining the operational semantics, focusing on representing machine state and auxiliary definitions.
2021-12-20	2022-01-02	Break for Christmas and New Year's.
2022-01-03	2022-01-16	Finish the definition of the operational semantics. Prove some useful lemmas regarding manipulating the structure of blocks of assembly.
2022-01-17	2022-01-30	Define the axiomatic semantics in the form of Hoare triples.
2022-01-31	2022-02-13	Prove and equivalence between the operational and axiomatic semantics.
2022-02-14	2022-02-27	Progress review. Begin formal verification of Barrett reduction.
2022-03-14	2022-03-27	Formally verify an implementation of Montgomery multiplication.
2022-03-14	2022-03-27	Begin work to formally verify one iteration of the number-theoretic transform, which is a form of discrete Fourier transform in the domain of integers modulo a prime.
2022-03-28	2022-04-10	Finish verification of one iteration of the number-theoretic transform.
2022-04-11	2022-04-24	Write-up.
2022-04-25	2022-05-08	Write-up.
2022-05-09	2022-05-22	Contingency weeks.
2022-05-23	2022-05-27	Submission. Begin work on presentation.