Static Resource Analysis of Smart Contracts

Nick Roberts
Carnegie Mellon University
nroberts@andrew.cmu.edu

November 3, 2017

1 Project Webpage

You may find documents related to my project at `www.andrew.cmu.edu/user/nroberts/400`.

2 Project Description

For my 15-400 project, I will be working to automatically infer the resource usage of smart contracts running on the Ethereum Virtual Machine (EVM). As a means to do this, I will extend the OCaml compiler to output EVM bytecode instead of OCaml bytecode. My research advisor will be Professor Jan Hoffmann (`www.cs.cmu.edu/~janh`), and I will be working with his postdoctoral student, Van Ngo Chan (`www.channgo.2203.github.io/`), as well.

2.1 Background

To talk about smart contracts and the EVM, it is first important to understand what the blockchain is. At a high level, a blockchain is a public ledger recording all transactions that have taken place in a system. Each participant in the network maintains their own copy of the ledger. Under cryptographic hardness assumptions, as long as a malicious agent does not control the majority of computing power in the network, items on the ledger cannot be forged. This level of security means that the blockchain is a suitable environment for *cryptocurrencies*, which platforms like Bitcoin and Ethereum realize in full.

Bitcoin only supports transactions involving its associated currency, bitcoin; likewise, the only state stored for an account is its balance. Unlike Bitcoin, Ethereum supports a generalized notion of transactions and account state: the ledger stores in each account code that may be triggered by transactions—this code is known as a *smart contract*. In its execution, a smart contract can modify its address space or trigger other smart contracts.

Static resource analysis is desirable because Etheurem's cost model corresponds to its associated cryptocurrency: Ether. Each bytecode instruction on the EVM is specified to have a certain *gas cost*, which the user pays for in Ether at the market rate. Since EVM bytecode is Turing-complete, transactions must give an upper limit on gas usage, exceeding which the computation will fail. As such, a user would want to know the resource usage of a smart contract *a priori*, especially if that contract was written by another user.

2.2 Approach

My work will involve two main components: an compiler from the core OCaml language to EVM bytecode, and a mechanism for inferring the resource usage of basic blocks at some intermediate stage of the compiler. In practice, the compiler will be an extension to the existing OCaml bytecode compiler[1]. Since EVM bytecode provides primitive instructions that do not correspond directly to OCaml language constructs, it will be necessary to design an interface to

[1]`https://github.com/ocaml/ocaml`

these primitives from the OCaml code. I will first accomplish this with a module with bindings for the specific bytecode instructions, but we will work toward a high-level library in the style of functional programming. As a basis for how this design was accomplished in another language, I will consider the design of the Solidity language[2], which compiles from a Java-like language to EVM bytecode.

Once I have successfully modified the OCaml compiler to output EVM bytecode, we will proceed by instrumenting a phase of the compiler with gas cost of basic blocks of bytecode. Given these annotations, we can apply techniques from the field of static resource analysis to extract a whole-program upper bound on gas cost.

3 Project Goals

3.1 75% Goal

A working compiler to EVM bytecode for the core OCaml language, not including higher-order functions. A low-level module with calls that map directly to EVM primitives. At least one smart contract on the Ethereum blockchain. No static resource analysis.

3.2 100% Goal

A compiler to EVM bytecode for a larger subset of the core OCaml language, including higher-order functions. A high-level module that abstracts away the EVM primitives, but that still compiles to a reasonably efficient program. At least one smart contract on the Ethereum blockchain. Static analysis of gas cost of basic blocks.

3.3 125% Goal

The above goal, in addition to a whole-program analysis of the gas cost.

3.4 Measuring Success

At the compilation level, my level of success corresponds to how fullly-featured a subset of OCaml I am able to compile to EVM bytecode. At the static analysis level, my level of success corresponds to the tightness of the upper bounds on gas cost.

4 Milestones

4.1 First Technical Milestone

This semester, I plan to become identify and become familiar with relevant parts of the OCaml and Solidity compilers so that I understand what modifications will be necessary to the base OCaml compiler to output EVM bytecode. I will write a compiler for an extremely restricted subset of the OCaml language: `let`-bindings and binary operators.

4.2 Future Milestones

- **January 31st**. Write module for direct access to EVM primitives from OCaml. Compile all arithmetic expressions from OCaml to EVM, not including user-defined data types.

[2] `https://solidity.readthedocs.io/en/develop`

- **February 14th**. Compile procedure calls to EVM, including the primitive operations exposed by the module, but excepting higher-order functions.
- **February 28th**. Compile user-defined data types from OCaml bytecode to EVM.
- **March 21st**. Compile higher-order functions to EVM bytecode.
- **April 4th**. Establish a mechanism for determining gas cost of basic blocks at some phase in the compiler. Infer the gas cost of these blocks.
- **April 18th**. Extend basic block gas cost to whole-program gas cost. Run a smart contract on the Ethereum blockchain.
- **May 2nd**. Evaluate performance of gas-inference mechanism.

5 Literature

5.1 On Ethereum and the EVM

- G. Wood. Ethereum: A Secure Decentralised Generalised Transaction Ledger (EIP-150 Revision). 2014. `https://github.com/ethereum/yellowpaper`.

5.2 On OCaml's Compiler

- X. Leroy. The ZINC experiment, an economical implementation of the ML language. Technical report 117, INRIA, 1990.
- The source code, linked as an earlier footnote.

5.3 On Static Resource Analysis

- J. Hoffmann, K. Aehlig, and M. Hofmann. Multivariate Amortized Resource Analysis. In *Proceedings of the 38th ACM SIGPLAN Symposium on Principles of Programming Languages*, pages 357–370, January 2011.

6 Resources

In addition to the OCaml compiler I linked earlier in this proposal, I will install the Solidity language and related tools. Solidity will be a useful reference as another compiler's approach to creating EVM bytecode. At a later point in the project, I will purchase Ether to run small smart contracts on the blockchain, but this will not be a large investment.