Static Resource Analysis of Smart Contracts – Milestone 5

Nick Roberts
Carnegie Mellon University
`nroberts@andrew.cmu.edu`
March 22, 2018

1 Summary of updates

In previous milestones, I focused my work on translations between Caml bytecode and EVM bytecode. For this milestone, I instead focused on facilities intended to support the typical blockchain developer, including a more sensible build system, access to EVM primitives like cross-contract calls and 256-bit arithmetic, and compatibility with protocols used by Solidity, and existing language for Ethereum. I have finished some of these components (the build system; access to primitives; arithmetic) but continue to work on others (cross-contract calls and compatibility with Solidity).

2 Accomplishments

Since last milestone, I have:

- Fixed several bugs in the compiler from Caml bytecode to EVM bytecode (henceforth the "EVM compiler").
- Added features to the EVM compiler, including records, recursive functions, and user-defined datatypes.
- Relocated the source code of the EVM compiler to a subproject of the Caml compiler so that it can make use of intermediate representations.
- Provided access for the programmer to many primitives offered by the EVM, including 256-bit arithmetic (which is not supported by the standard OCaml compiler) and useful bytecode instructions (like determining the remaining gas in the transaction).
- Decided on a design for the "phase distinction" between code that runs upon contract creation and code that runs upon a contract being called.
- Researched the Solidity ABI for performing calls between contracts.

2.1 Meeting the milestone

My goal as stated in the previous milestone document was to add more language features to the EVM compiler and to decide on a design for the "phase distinction" mentioned in the above section. I've accomplished (more than) both of these goals.

2.2 Surprises

After discussing aspects of the blockchain with Michael Coblenz, a Ph.D. student in the POP group, I became aware of some misunderstandings I had about the semantics of Solidity (the most-used language for Ethereum). I had believed that Solidity classes were the objects placed on the blockchain as smart contracts, but it is in fact *instances* of Solidity classes. I will consider whether a similar sort of semantics is desirable for my compiler (but this will require some thought, since the compiler only supports the Caml part of OCaml right now, and so it would be non-trivial to handle object-oriented concepts such as instances of objects).

3 Looking ahead

3.1 Revisions to future milestones

For next milestone, I am focusing extensively on understanding Solidity's ABI interface for external function calls so I can adopt this protocol into my own compiler as both the caller and the callee. By next milestone, I wish to have external function calls fully implemented.

This is not something that I anticipated in my original proposal, and it is forseeable that the static resource analysis component of my project will be relegated to a minor discussion (or something left to future work). I still hope to address it, but the usability of my project as a legitimate compiler remains for now as the highest priority.

3.2 Resources needed

There are no further resources needed at this time.