Static Resource Analysis of Smart Contracts – Milestone 4

Nick Roberts
Carnegie Mellon University
nroberts@andrew.cmu.edu
February 28, 2018

1 Summary of updates

For this milestone, I worked on a compiler from Caml bytecode to Ethereum Virtual Machine (EVM) bytecode. I have succeeded in compiling a significant fragment of OCaml's core language, including arithmetic, function application, and closure creation. For future weeks, I will work on extending the supported fragment of OCaml's language and designing a framework for programmers to write code to be executed in separate phases (upon contract creation vs. upon calling a function in the contract.)

You can track my progress with the compiler on Github.[1]

2 Accomplishments

Since last milestone, I have written a compiler from a fragment of OCaml's core language to the EVM. This compiler supports the following constructs:

- Arithmetic
- Stack-allocated variables
- Control flow operators, like `if` and `while`
- Function application
- Closure allocation on the heap

To automate testing of the emitted bytecode, I wrote a script that spins up a local instance of the Ethereum blockchain (using `ganache-cli`[2]), creates an instance of a smart contract out of the provided bytecode, and calls the contract. I have fixed all bugs that testing has detected so far.

2.1 Meeting the milestone

I have met the goal set in the previous milestone, which was to compile a large subset of OCaml code to the EVM.

2.2 Surprises

There are headaches on both side: the Caml bytecode and the EVM bytecode each have their own eyebrow-raising peculiarities. For example, neither uses labels for jumps (Caml uses relative offsets and the EVM uses absolute offsets), but the length of the encoding of the offset may affect the offsets of later instructions in the EVM. So far, I have been able to resolve these problems, but it is nice to let off some steam in this section.

[1] `https://www.github.com/ncik-roberts/ocaml-interpreter`
[2] `https://github.com/trufflesuite/ganache-cli`

3 Looking ahead

3.1 Revisions to future milestones

For next week, I am working on developing a sensible protocol for users wishing to write a program using our compiler. For example, with the EVM, some code can be executed during contract creation, and some code can be executed every time a contract is called. It is desirable to make this phase distinction explicit in the OCaml programs accepted by my compiler. This did not appear in my original proposal, but it is an important aspect I want to consider.

I will continue to add support for more features in the compiler. Adding the feature proposed in my original project proposal for this date (user-defined datatypes) will be relatively trivial, since the Caml bytecode is a mostly-simple calculus.

3.2 Resources needed

The cost of placing a small contract on the blockchain can run rather high. In the current incarnation of my compiler, each line of Caml code costs approximately $0.25, but this may be decreased as I optimize the compiler. Dr. Hoffmann has indicated that he has a Ph.D. student interested in using my compiler to write a basic program on the blockchain, so we will end up having to pay for a medium-sized smart contract. (He has also indicated that this won't be a problem.)