IVY: A PROGRAMMING LANGUAGE TARGETING EVM

Yigit Ozkavci
Advisor: Can Ozturan

Introduction

Ivy is a programming language designed for Ethereum Virtual Machine. Ivy enables us to write smart contracts and deploy them directly to an Ethereum blockchain.

Motivation

Smart contracts are widely accepted concept of transferring values without the existence of a middleman. A smart contract lives in the blockchain like any other entity, and is able to receive & send valuables, with the decision being made via contract's internal state.

Every computation on the smart contracts costs gas, a measure for computational power required by the contract; this makes the content of smart contracts even more critical since the efficiency of computation directly affects the costs of executing the contract, in real money.

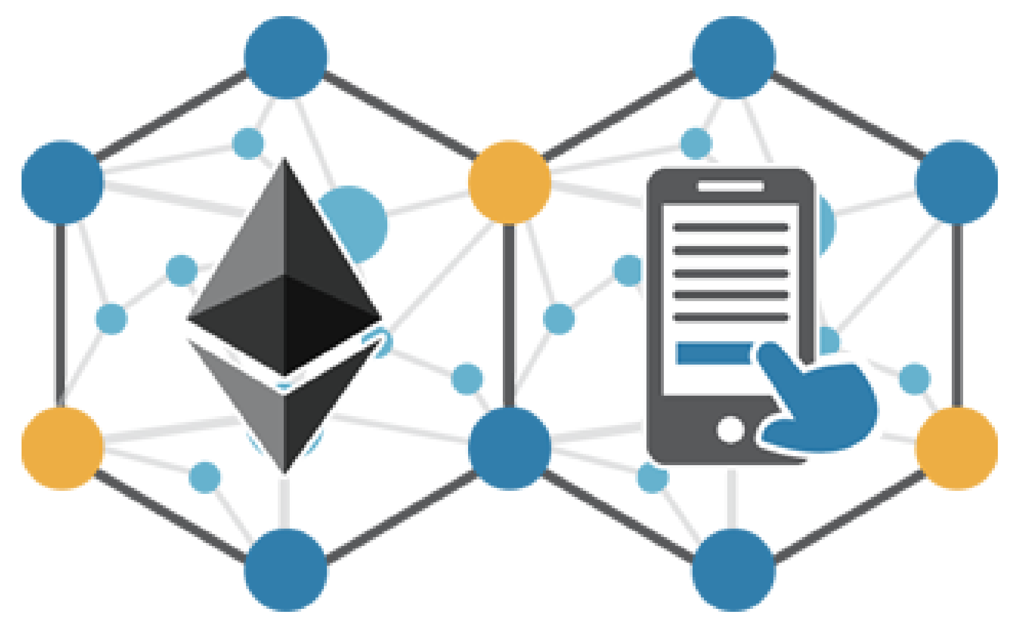

Figure 2: Smart Contract

Since this is the second term and we continued developing Ivy, which was already a mature programming language, we've made crucial improvements in order to make Ivy more convenient and powerful.

Language Features

Features Introduced First Term:

- Primitives: int, char, bool
- Array types: int[], char[], bool[]
- Branching (if-else statements)
- Loops
- Functions
- Arithmetic (+,-,*,/) and comparison (>,==,<) operators
- Type-checking
- Parsing error handling
- Compile-time error handling

Features Introduced Second Term:

- Persistent contract storage (corresponding to contract variables in Solidity)
- Runtime heap-space arrays
- Hashmaps (can be persistent and temporary) (for security considerations, see [2])
- Deployment (to any Ethereum blockchain network)
- Runtime Logging (or events, in Solidity convention)
- Runtime Exceptions (by halting the transaction)
- Syntax support for resizing and allocating dynamic arrays

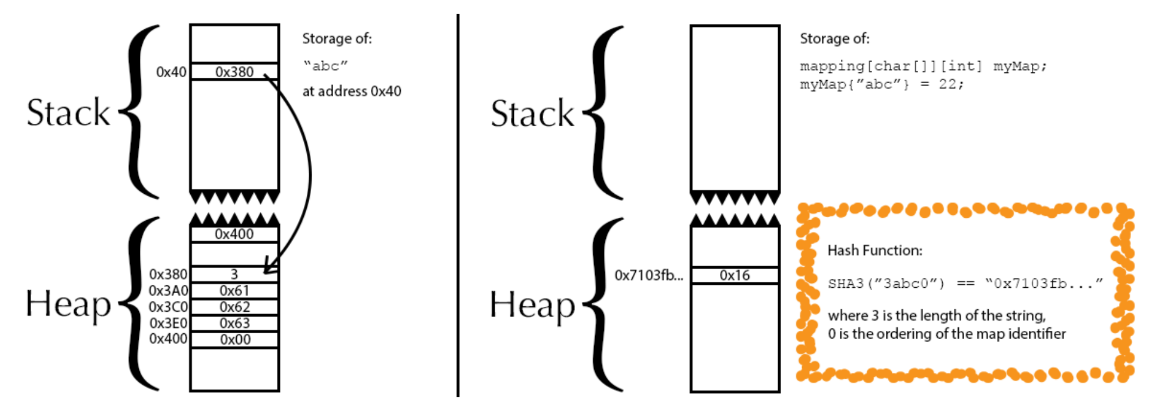

Figure 3: Ivy Memory Architecture

Development

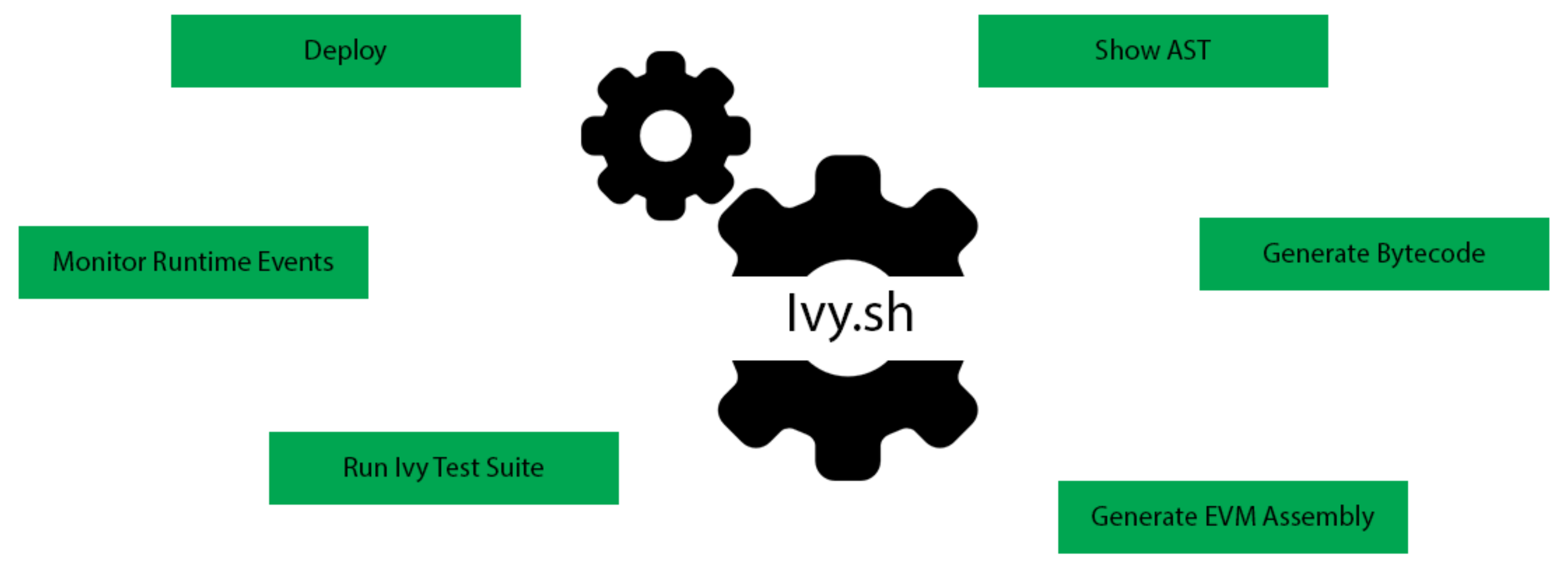

Figure 1: Ivy.sh

Ivy has grown over time and in order to satisfy its needs, we built a compiler program, which can be used to perform several tasks other than just compiling Ivy.

Currently Ivy has 13 test scenarios which we perform in real testing network, collect the results and assert to generate a testing report. Every patch to Ivy compiler should satisfy the test requirements and pass all the scenarios.

Research

The biggest challenge in this term was to figure out how to represent runtime data structures in the context of EVM. In order to do that, we studied yellowpaper[1] on how persistent memory works, and how one handles runtime exceptions.

In EVM, exception means halting the execution and returning 0 as the result of the operation; to debug such cases, we use $LOG0, LOG1, \ldots$ instructions of the EVM to provide a meaningful message with the contents of a portion of the memory, and lastly a topic to categorise the log message.

Firstly, we separated our memory into two spaces: stack and heap. Stack, we know every address on, of which size is determined at compile time. By exploiting lazy evaluation of Haskell, we can use the address where stack ends and heap begins, before even computing it (details on this will be in the report). For heap, we only know where it begins, and everytime we expand heap we update the last address on the heap to help us in allocating future values.

We have the convention of storing arrays in the format: length, first elem, second elem, ...; this convention helps us while traversing or resizing or accessing a single element of the array.

For hashmaps, addressing is a bit different; our hashing function exploits the fact that EVM owns a virtual memory of size 2^{256}. We use the hashing pattern that also Solidity uses: we concatenate key with the order of the mapping identifier, then take the keccak256 hash of it to compute its address. This operation is done in runtime (otherwise we could not use keys with dynamic types such as arrays) with the instruction $SHA3$.

References

[1] DR. Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger.

[2] Derek Anderson. Are ethereum contracts vulnerable to hash table poisoning attacks?

[3] Elm programming language: http://elm lang.org/.

[4] Purescript programming language: http://www.purescript.org/.

[5] Lisp Like Language (LLL): http://lll docs.readthedocs.io/en/latest/.

Future Research

Over the time of 1 year, Ivy has grown into a usable and convenient programming language for writing Ethereum smart contracts. We plan on giving Ivy a position of an IR (Intermediate Representation) to give rise to a pure functional, high level programming language.

Ivy currently lacks necessary abstractions for writing safe and efficient smart contracts, but also low-level enough to be used as a base language to compile to, just like LLL[5].

Contact Information

Web https://github.com/yigitozkavci
Email yigitozkavci8@gmail.com
Phone +90 535 084 76 02