Interactive theorem proving in the Ethereum project

Yoichi Hirai

Ethereum Foundation

Munich, May 31, 2017

Outline

Outline

Ethereum: Program Execution without Trusted Admin

Computers have owners and administrators.

- Will my program be executed unmodified?
- Will my program be available?
- Will the state of my program be available & unmodified?

Your own machine might be OK. What about the server side?

Ethereum currently uses a Bitcoin-like approach

1. to replicate programs and program states, and
2. to agree on execution traces.

Over 20,000 nodes[1] are running a clone of the Ethereum Virtual Machine (EVM).

[1] According to `ethernodes.org`.

A Block: a Unit of History

$$\left.\begin{array}{c} (s_0, t_0) \overset{f}{\mapsto} s_1 \\ (s_1, t_1) \overset{f}{\mapsto} s_2 \\ \vdots \\ (s_{n-1}, t_{n-1}) \overset{f}{\mapsto} s_n \end{array}\right\}$$

a block contains $t_0, \ldots, t_{n-1}, \texttt{KECCAK}(s_n)$ and the hash of the previous block.

f Ethereum Virtual Machine (with hidden params: current time etc.)

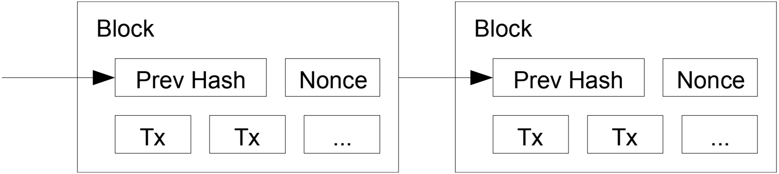
t_i i-th input (called transaction) in a block

s_i machine state before i-th input

- f is deterministic and total.
- f is implemented in C++, Python, Go and Rust.
- Any diff between implementations appears in the hash.

Choosing a History by Proof-of-Work

A block can be sealed only with a nonce found with luck, electricity and GPU; the hash of the block has to be small.



(Picture from Nakamoto[2])

If most new blocks follow the longest chain, a history emerges.

In Ethereum,

- a block is sealed every $\approx$15 sec. ($\leftrightarrow$ Bitcoin 10 min.)
- blocks include uncles. Not longest but heaviest is sought.
- proof-of-work is not considered secure enough; proof-of-stake protocols are being designed.

[2] `https://bitcoin.org/bitcoin.pdf`

EVM State: a Balance Sheet with Code

The machine has a map from 2^{160} to accounts.

An account contains:

- balance: 2^{256}
- nonce: 2^{256}
- storage: $2^{256} \to 2^{256}$
- code: a list of bytes
  (which controls the balance and the storage)

Empty accounts are very compactly represented.

The balance is denominated in 10^{-18} ETH:

- fee for running Ethereum Virtual Machine (EVM)
- reward for authoring blocks

EVM Transaction

A private key holder of an address can spend the balance and execute a transaction in the EVM.

The spent balance goes to the author of the block.

The private key holder can:

deploy code to an address (determined by a hash function)

call code on an address with ETH transfer.

The called code can further call codes.

Typical Ethereum Usage: Deposits & Announcements

Ethereum Name Service is a sealed second-price auction.
The price is locked while the name is held.
Roughly 66,000 ETH ($\approx$ 12,000,000 EUR) locked for 33,000 names.

Voting Protocol McCorry, Shahandashti and Hao [FC 2017] implemented a voting protocol on Ethereum.
The protocol requires a public bulletin board; and uses deposit to incentivize participants to perform all steps.

The Famous Bug

"The DAO" (an investment club):
funds moved out much more quickly than expected
17% of total existing ETH affected.
Many miners[3] accepted a protocol change to remedy this particular case; the network split.

This does not work:

1. Develop the source code of Ethereum contracts on GitHub.
2. Enough people would look at it.
3. Bugs would be found early enough.

[3] Miners run GPUs to find a good nonce.

EVM might be a Good Formalization Target, I Thought

- Immutable code sounds crazy unless it's proven correct.
- Deterministic and total VM: no monads.
- Termination by decrementing gas: sounds familiar.
- A stack machine: no variable scopes.
- Multiple implementations are compared by tests;
  the formalization can be tested as another client.
- The specification is 32-page-long.

I was in Dresden with a 3,000-page-long Intel manual.

Outline

How the Paper Spec and Lem Spec Look

The EVM definition in Lem has 2,000 lines.
Most instructions are simply encoded as functions in Lem:

Yellow Paper (original spec):

0x06	MOD	2	1	Modulo remainder operation.
				$\boldsymbol{\mu}'_{\mathbf{s}}[0] \equiv \begin{cases} 0 & \text{if} \quad \boldsymbol{\mu}_{\mathbf{s}}[1] = 0 \\ \boldsymbol{\mu}_{\mathbf{s}}[0] \bmod \boldsymbol{\mu}_{\mathbf{s}}[1] & \text{otherwise} \end{cases}$

Lem:

```
| Arith MOD -> stack_2_1_op v c
      (fun a divisor -> (if divisor = 0 then 0 else
            word256FromInteger ((uint a) mod (uint divisor))
      ))
```

Overall Data Structure

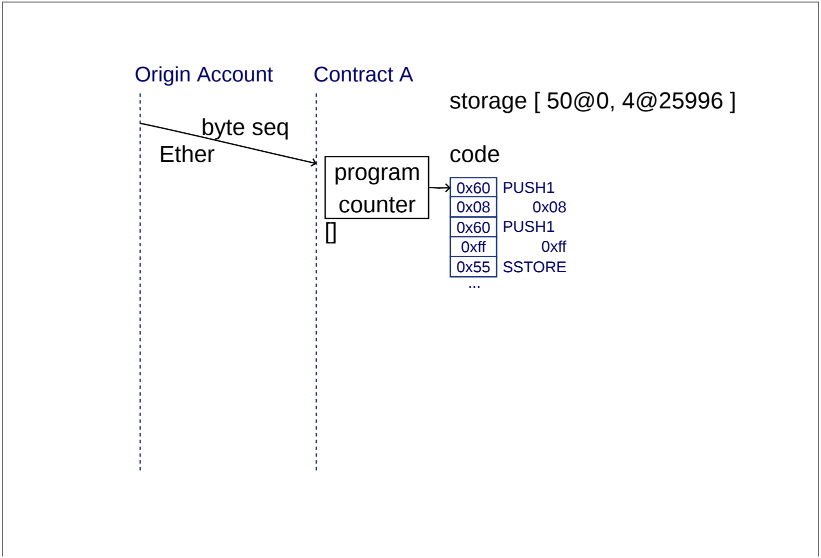
An account contains:

- balance (256-bit word)
- code (byte sequence)
- storage (2^{256} words)
- nonce (256-bit word)

A contract invocation provides:

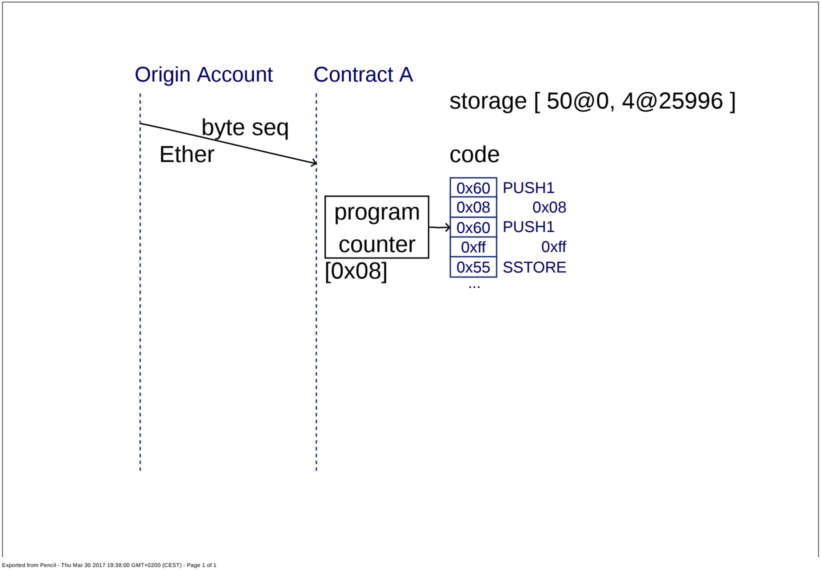
- input data (byte sequence)
- memory (2^{256} bytes, charged by max accessed word)
- stack (up to 1024 words)
- information by miner (timestamp, block number etc.)
- information by caller (transferred ETH, gas limit etc.)

How EVM Works 1

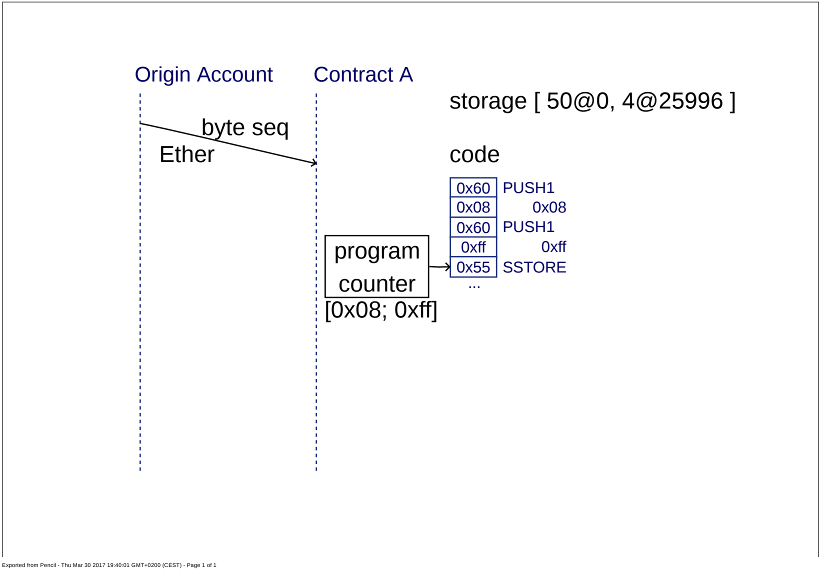


Exported from Pencil - Thu Mar 30 2017 19:37:34 GMT+0200 (CEST) - Page 1 of 1

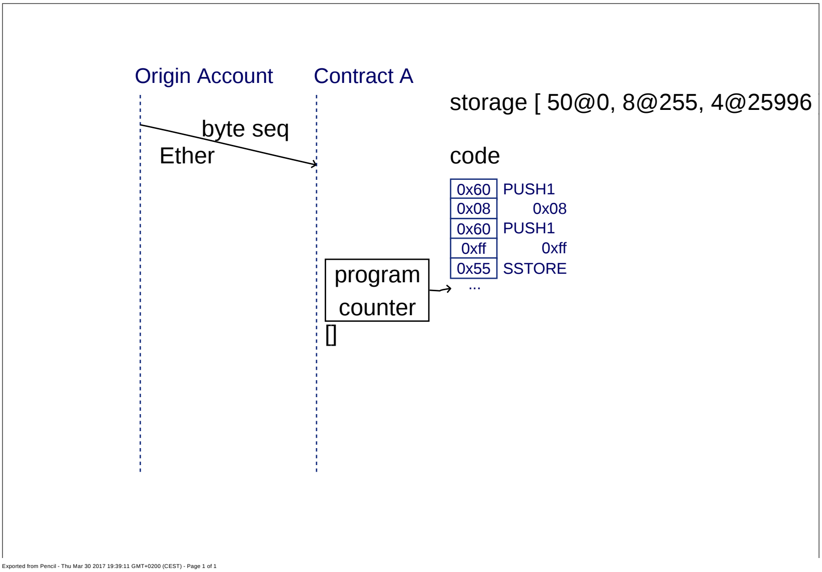
How EVM Works 2



How EVM Works 3



How EVM Works 4



Program Syntax

A list of bytes (one byte = eight bits).

- Most instructions consist of one byte.
- Only `PUSH1`, `PUSH2`, . . . , `PUSH32` instructions contain multiple bytes.

`PUSH4 0xaabbccdd` is represented by `0x63aabbccdd`.

Always parse from the beginning!

Stack

- list of words (a word is an untyped 256-bit)
- empty when a program starts
- each instruction takes&puts constant numbers of elements

$$[0, 1, 2, 3]$$
$$\downarrow \quad \text{ADD}$$
$$[0, 1, 5]$$

- throws when number of elements < 0 or ≥ 1025
  An exception reverts the state changes in the current call but depletes all gas.

Control Flow

(I talk about the VM as if it works as time progresses.)
The first instruction in the program is executed first.
The next instruction in the program is usually executed next, but

0x56 JUMP jumps to the position specified by the topmost stack element

0x57 JUMPI jumps if the second stack element is non-zero

A jump goes to a 0x5b JUMPDEST or throws.
(JUMPI, when the condition is false,
does not check the validity of jump destination.)

Cannot determine stack layout at each instruction $\rightsquigarrow$ EVM 1.5

Memory

A mapping $2^{256} \rightarrow 2^8$ (a word-addressed byte array)

- contains zeros when a program starts
- 0x51 MLOAD copies a word from the memory to stack
- 0x52 MSTORE moves a word from the stack to memory
- 0x53 MSTORE8 moves a byte from the stack to memory

The memory is byte-addressed but word-accessed; requiring concatenation and splits somewhere.

An address overflow is silently taken modulo 2^{256}.

Storage

A mapping $2^{256} \to 2^{256}$ (a word-indexed word array)

- configurable contents at the beginning of a blockchain (zero by default)
- shared by all code execution on the same account
- 0x54 SLOAD copies a word from storage to stack
- 0x55 SSTORE moves a word from stack to storage

Expensive (around 8 cents per SSTORE).

Given a machine state, an address $\in 2^{256}$ and an index $\in 2^{256}$, a storage content $\in 2^{256}$ is found.

Informational Instructions

0x30 ADDRESS shows the currently executing address

0x31 BALANCE reveals the balance of any address

0x33 CALLER

0x34 CALLVALUE is the amount of ETH sent from the caller

0x37 CALLDATACOPY

0x3c EXTCODECOPY copies any address's code into memory

0x42 TIMESTAMP current time according to the block author

0x43 NUMBER the number of blocks so far

0x44 DIFFICULTY how much luck$\times$energy to seal a block

⋮

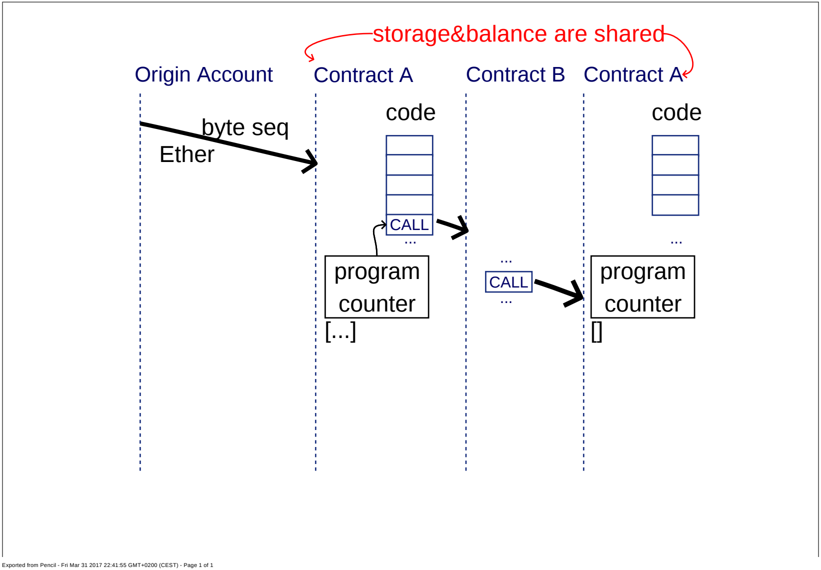
Calls Execute Code

A private key holder can call an Ethereum contract (account with non-empty code).

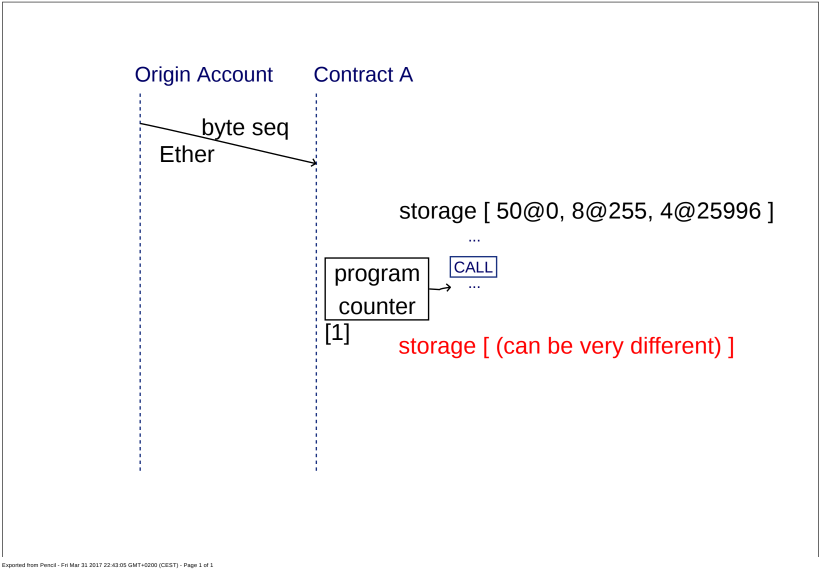
A contract can call a contract.
CALL instruction takes seven words from the stack.

- amount of gas shared with the callee
- destination of the call
- amount of transferred ETH
- input data (offset + size on the memory)
- memory region for receiving output data

An Annoying Phenomenon Called Reentrancy (Transaction's View)



An Annoying Phenomenon Called Reentrancy (Invocation's View)



Outline

Properties Wanted about a Contract

Safety Properties

- only this kind of callers can alter storage
- only this kind of callers can decrease the balance[4]
- the invalid opcode `0xfd` is never hit
  (Some compilers encode safety properties using 0xfd)

Game Theoretic / Cryptographic Properties

In second-price sealed auctions, “bidding honestly” should be the dominant strategy.
Does this property carry over to the implementation?

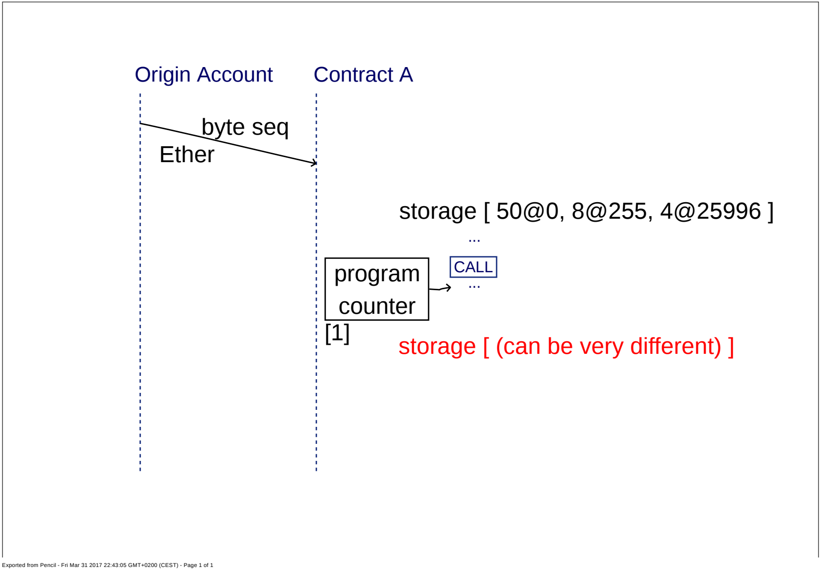
[4] Anyone can add balance to any account ☹

Phases of EVM Modeling

Phase 1	single call—done
Phase 2	caller-callee interaction—in testing & debugging
Phase 3	follow the blockchain—not started

Phase 1: Take the Single Invocation's View

Involves some artificial nondeterminism.



Special Treatment of CALL

During CALL instruction, nested calls can enter our program.
Nasty effects after executing CALL:

- the balance of the contract might have changed
- the storage of the contract might have changed

Our blackbox treatment of CALL during phase 1

- by default, the storage and the balance change arbitrarily during a CALL.
- optionally, you can impose an invariant of the contract, which is assumed to be kept during a CALL
  but you are supposed to prove the invariant.

Outline

Lem

- a specification language
- translates into HOL4, Isabelle/HOL, OCaml (and Coq)

How I started using Lem

1. I started this project in 2015 in Coq.
2. I tried Isabelle/HOL and my proofs got shorter.
3. Sami Mäkelä saw this and started the Lem version.

OCaml for Testing

- Lem to OCaml extraction
- OCaml code to parse test cases (simplest “VMTest” format)

Need to run other formats

- GeneralStateTest format: new tests are in this format
- BlockchainTest format
- Follow the Ethereum history

I’m in an optimal position to do these.

Isabelle/HOL for Proving

Lem to Isabelle/HOL seems to be working.

As an off-the-shelf symbolic executor

A huge apply-script.
One instruction takes 15 seconds for a realistic code (storing the program in an AVL tree, and optimize simplification).

Some separation logic

A predicate for each byte in the memory, each word in storage&stack.
I'm instantiating frame rules explicitly; which feels wrong.
(I heard about separation algebra, which I should learn.)

I Received a Challenge: Original Text

Message types

- commit(HASH, view)
- prepare(HASH, view, view_source),
  -1 $\leq$ view_source < view

Slashing conditions

1. commit(H, v) REQUIRES 2/3 prepare(H, v, vs) for some consistent vs
2. prepare(H, v, vs) REQUIRES 2/3 prepare(H_anc, vs, vs') for some consistent vs', where H_anc is a (v-vs)-degree ancestor of H, UNLESS vs = -1
3. commit(H, v) + prepare(H, w, u) ILLEGAL if u < v < w
4. prepare(X1, v, vs1) + prepare(X2, v, vs2) ILLEGAL unless X1 = X2 and vs1 = vs2

Challenge Continued

Accountable safety argument

(proof path - assume two incompatible values got committed, show 1/3+ SLASHED)

Case 1

2/3 commit(X, v) & 2/3 commit(Y, v)
→ 2/3 prepare(X, v, vs) & 2/3 prepare(Y, v, vs') (1.)
→ 1/3 SLASHED (4.)

Case 2

2/3 commit(Y, v2) & 2/3 commit(X, v1),
Y is NOT a (v2-v1)-degree descendant of X, define Y[i] to be the ancestor of Y in view i
→ 2/3 prepare(Y[v2], v2, vs), vs < v2 (1.)
→ 2/3 prepare(Y[vs], vs, vs') (2.)
→ ...
[continue induction until vs' < v1]
(Two base cases follow.)

Turned out terse but followable in Isabelle/HOL (1,800 lines).

Outline

We Tested Our EVM Definition (Phase 1) against Implementations' Common Test

- Luckily, we have test suites for EVM definitions
  - The test suites compare Ethereum Virtual Machine implementations in Python, Go, Rust, C++, . . .
  - All EVM implementations need to behave the same, lest the Ethereum network forks (ugly)
- Definitions in Lem are translated into OCaml
- Our OCaml test harness reads test cases from Json, runs the Lem-defined EVM, checks the result v.s. expectations in Json
- VM Test suite: 40,617 cases (24 cases skipped; they involve multiple calls)

Problems in LaTeX Specification

Test suits are the spec in effect; the LaTeX spec is not tested.
While writing definitions in Lem (or previously in Coq)

- memory usage when accessing addresses $[2^{256} - 31, 1)$
- an instruction had a wrong number of arguments
- ambiguities in signed modulo:
  $\mathbf{sgn}(\mu_\mathbf{s}[0])|\mu_\mathbf{s}[0]| \bmod |\mu_\mathbf{s}[1]|$
- some instructions touched memory but did not charge for memory usage
- malformed definition: **o** was defined to be **o**

While testing the Lem definition:

- spurious modulo 2^{256} in read positions of call data
- exceptional halting did not consume all remaining gas

Proving Theorems about Ethereum Programs

We used Isabelle/HOL to prove theorems about Ethereum programs.
One theorem about a program (501 instructions) says:

- If the caller's address is not at the storage index 1, the call cannot decrease the balance
- On the same condition, the call cannot change the storage

Techniques:
Brute-force directly on the big-step semantics
(naïvely ignoring many techniques from 1960's and on).

- Human spends 3 days constructing the proof
- Machine spends 3 hours checking the proof

One Proving Strategy that We Took

1. Speculate an invariant of a contract
   "the code of the account can only stay the same or become empty"
2. Prove the invariant, assuming the invariant on reentrant calls
3. (hand-waiving argument that reentrant depth is finite)
4. Take the invariant for granted and prove pre-post conditions
   "if the caller is not the owner, the balance of the account does not decrease"

Sami Mäkelä defined a whole transaction:
the first step for removing the hand-waiving.

Outline

Watching and Applying Protocol Changes

Ethereum Foundation is a good place to see what's coming. (Discussions are usually public but finding the discussions is not always easy.)

The next change Metropolis is planned for this summer.

Ice age prompts protocol changes.

EIPs (Ethereum Improvement Proposals) change constantly.

Tests generated by C++client

All clients implement the same changes.

Yellow paper modifications are made by me.

EVM 1.5 and eWASM.

A Common Assumption: No Hash Collisions

Solidity (a popular language) encodes $w\colon 2^{160} \to 2^{160} \to 2^{256}$ in the storage $2^{256} \to 2^{256}$.

$w(x, y)$ is stored in the storage index
$\texttt{KECCAK}(y \cdot \texttt{KECCAK}(x \cdot \bar{w}))$
where $\bar{w}$ is a number reserved for w.

This is OK or we witness hash collisions. How to formalize this?

More Work

Ongoing

- testing the formalization of a whole transaction, containing transactions containing calls
- modular reasoning on bytecode snippets (Hoare logic w/ separating conjunction)

Not started

- implementing the next protocol change
- common Ethereum contract method/argument encoding
- connect to test/main network

Open

- semantics of Solidity (or a similarly approachable language)

Summary

- We defined EVM for proof assistants Isabelle/HOL, Coq and HOL4
- The EVM definition is somehow usable for proving Ethereum contracts for a specification

- Outlook
  - Testing efforts underway for phase 2.
  - Proof/tool/language/protocol developments in the proof assistants welcome
    `https://github.com/pirapira/eth-isabelle`
    (Apache License ver. 2 except material from Lem)