Blockchains: new home for proven-correct software

Paris, 2017-2-17

Yoichi Hirai
formal verification engineer,
the Ethereum Foundation

Lyon: 2014 January

“Have you heard of a web site where you can get <u>Bitcoin for proving theorems</u>?”

“Yeah,
I created the proof market.”

Proving software correct

- In Lyon, I was attending workshops about formal verification
- using interactive proof assistants

Coq

HOL

- they use only ~20 inference rules to derive the ~~whole~~ math
- … and that code matches specification

Formal in “formal verification”

Usual Math

idea

correct but boring

https://en.wikipedia.org/wiki/Emmy_Noether#/media/File:Noether.jpg

Formalised Math

no idea

just looking at “form”

correct

Use Proof Checkers against lots of cases

- Kepler’s conjecture “ is the most compact”
- need to see all other ways are less efficient
- This involves checking lots of corner cases.
  Flyspeck project (led by Thomas Hales) used
  Isabelle and HOL-light
- (That sounds useful for software.)

Proven-correct software

- seL4: a microkernel
  ARM assembly proven to behave as Haskell-like spec
  (NICTA, Australia, 2009)

- CompCert: a C compiler
  results proven to behave the same as the C source
  (INRIA, France, 2008; Xavier Leroy)

Compiler breaker could not break CompCert

found bugs in every tool that it has tested

more than 50 bugs in GCC,
more than 100 bugs in Clang

“The striking thing about our CompCert results is that the middle-end bugs we found in all other compilers are absent.”

http://www.cs.utah.edu/~regehr/papers/pldi11-preprint.pdf

Testing shows the presence, not the absence of bugs

—E.W. Dijkstr

Can proofs show absence of bugs?

No.

A bug = “what happens”

- <u>“what people think should happen”</u>

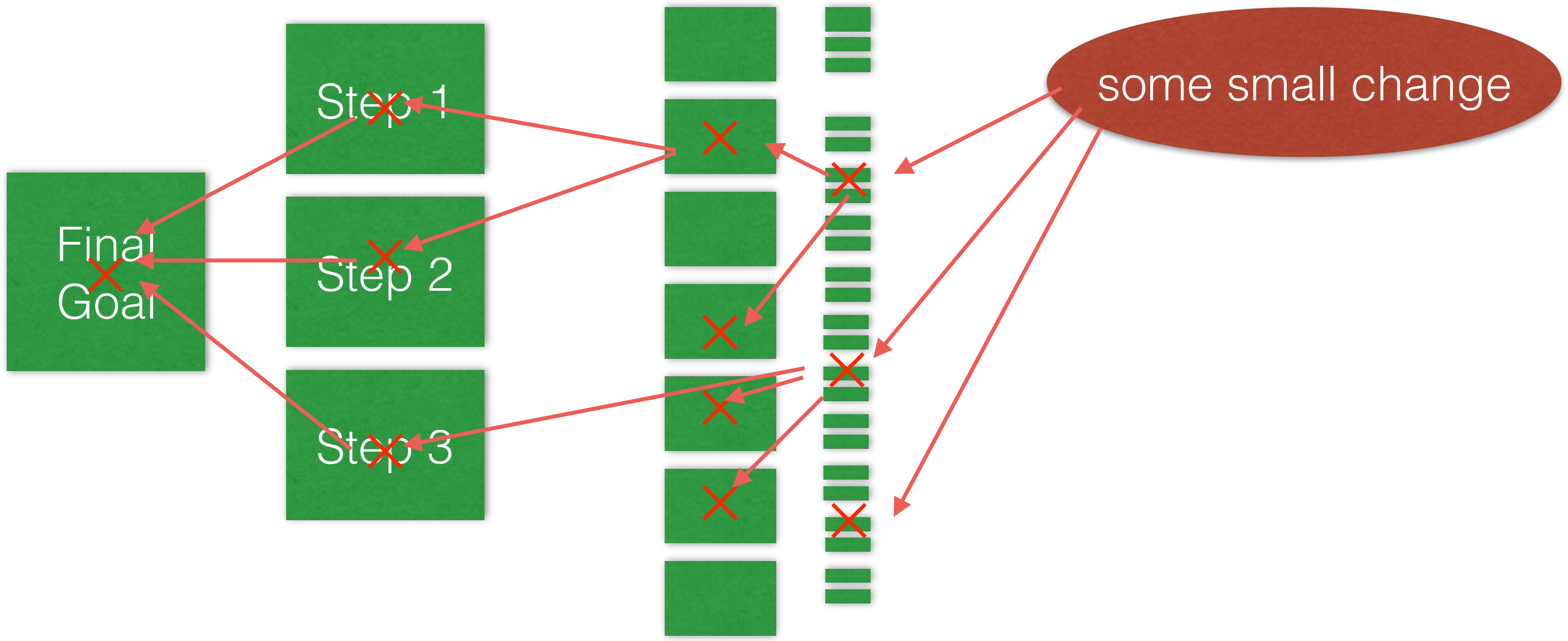
↑

not accessible, until people are surprised

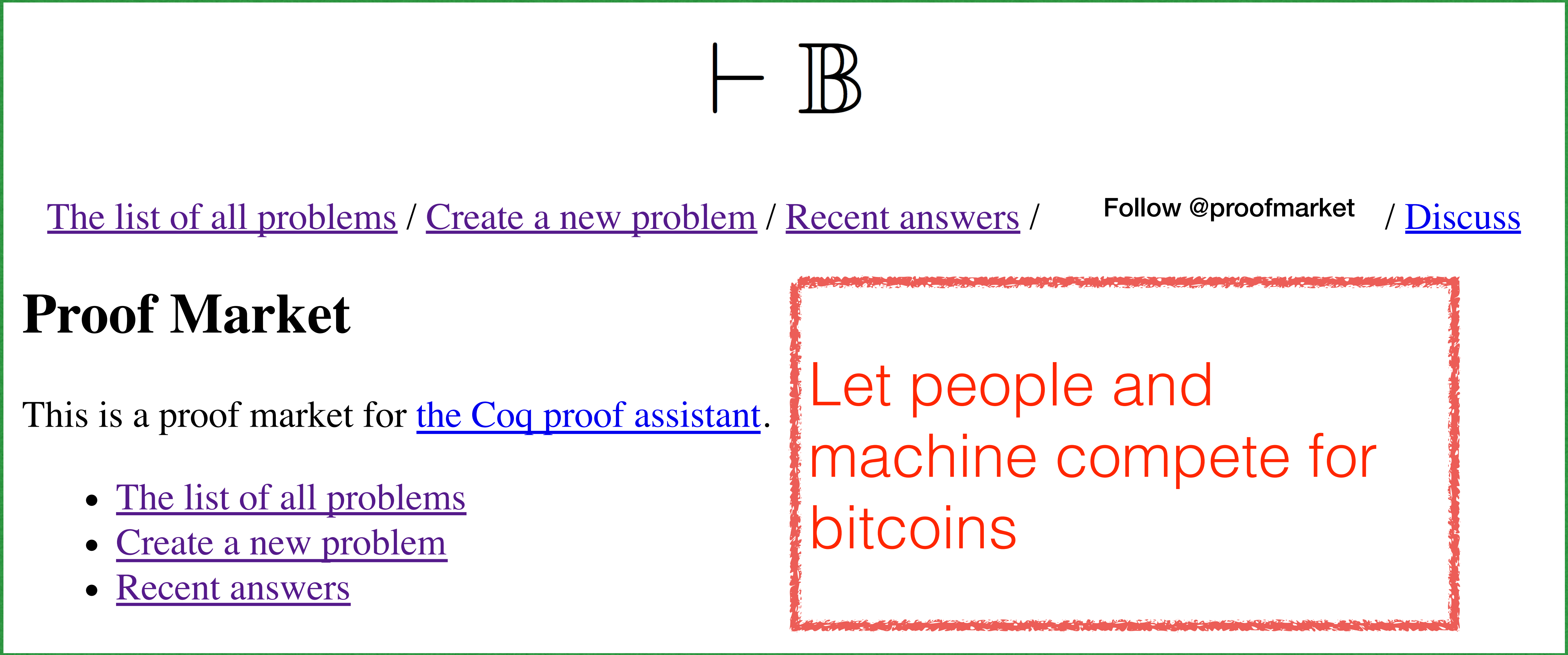
But, proofs can compare implementations and specifications, for all corner cases.

It takes time to prove software correct

CompCert took
  100,000 lines of Coq & 6 person-years of effort



That's why proof market



$\vdash \mathbb{B}$

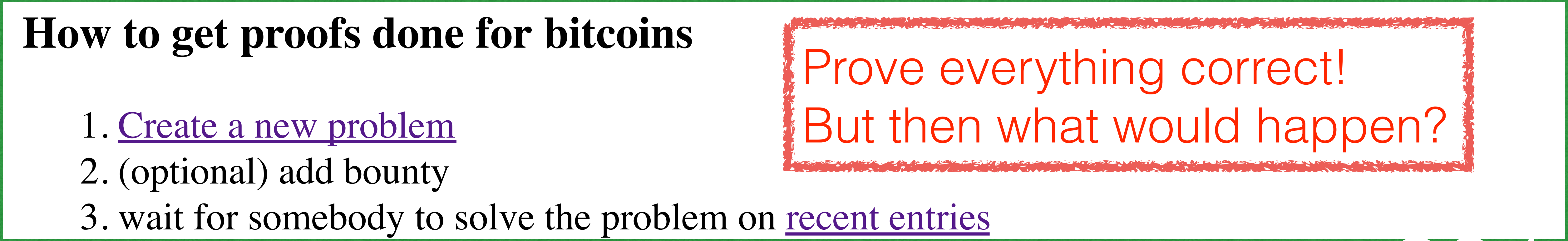
The list of all problems / Create a new problem / Recent answers / Follow @proofmarket / Discuss

Proof Market

This is a proof market for the Coq proof assistant.

- The list of all problems
- Create a new problem
- Recent answers

Let people and machine compete for bitcoins



How to get proofs done for bitcoins

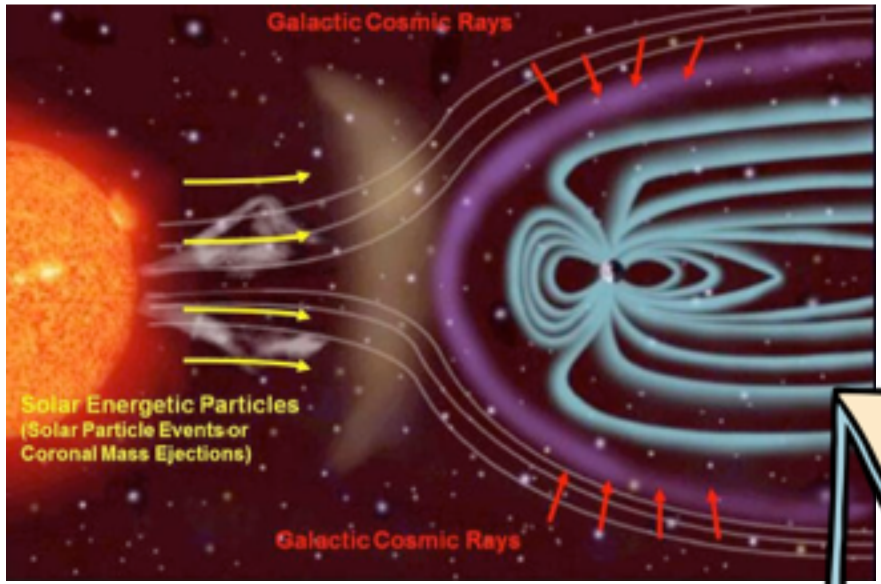
1. Create a new problem
2. (optional) add bounty
3. wait for somebody to solve the problem on recent entries

Prove everything correct!
But then what would happen?

2014

A cat can break proven-correct software

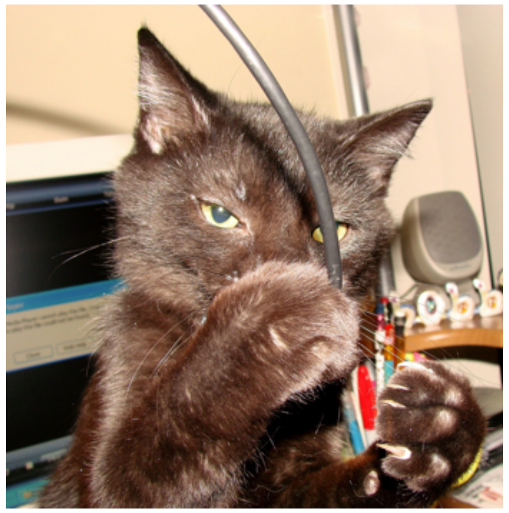
https://pixabay.com/en/cat-computer-cable-playing-animal-70736/

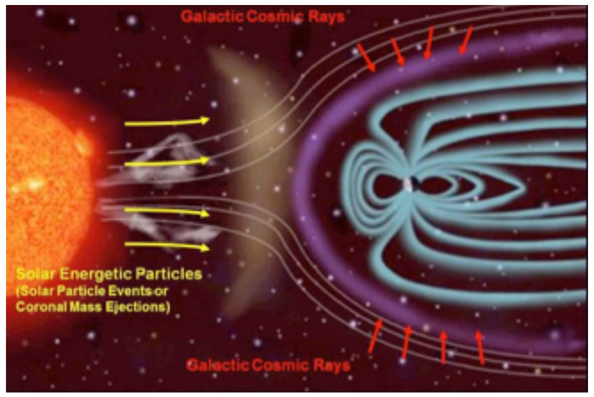


https://openclipart.org/detail/132427/penguin-admin

Ethereum against illogical failures

- cats-resistant
- radiation resistant
- bad admin resistant





No single cat can break a proven-correct smart contract?

https://static.pexels.com/photos/54632/cat-animal-eyes-grey-54632.jpeg

- I doubt it.
- But Ethereum seems an optimal deployment target for proven-correct software.

A simple theorem: ADD does addition

pre-condition

```
"triple {OutOfGas}
  (⟨ h ≤ 1023⟩ **
   stack_height (h + 2) **
   stack (h + 1) v **
   stack h w **
   program_counter k **
   gas_pred g **
   continuing
  )

  {(k, Arith ADD)}

  (stack_height (h + 1) **
   stack h (v + w) **
   program_counter (k + 1) **
   gas_pred (g - Gverylow) **
   continuing
  )"
```

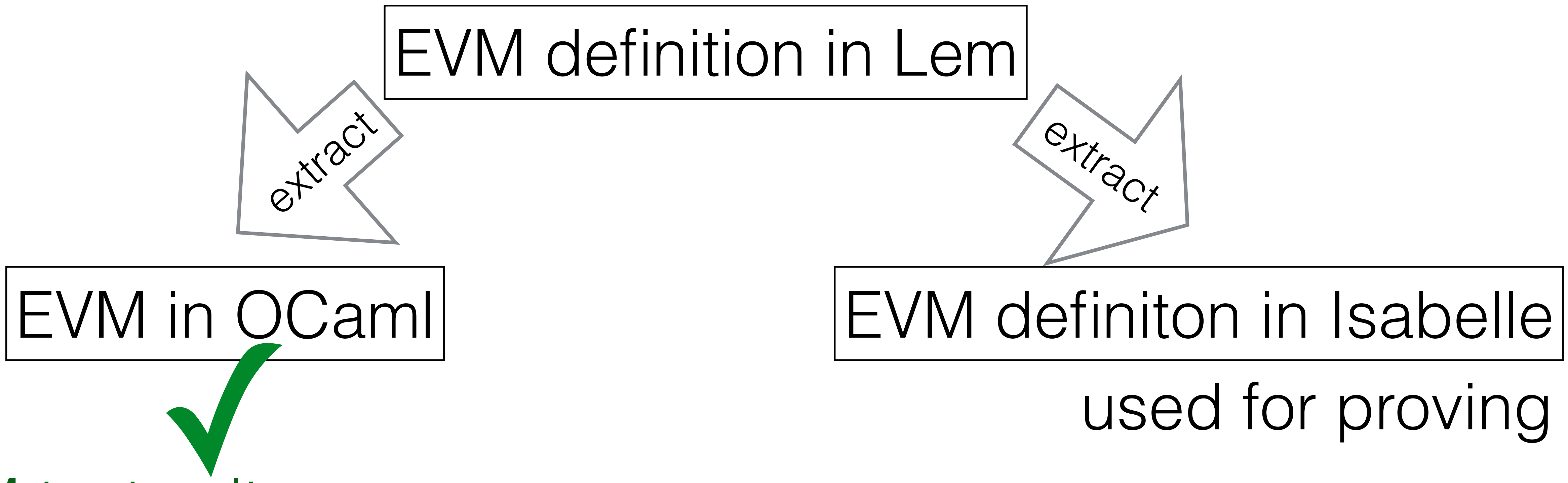
post-condition

Can that scale?

- Yes, but slowly
- Verified bytecode snippets can be composed
- It takes 5 minutes of manual work to combine two bytecode snippets
- ~ 10 instructions / hour
- more speed requires some months of tooling

Does this match the actual Ethereum Virtual Machine?

- Yes, as far as the VM test suite can tell.



VM test suite passes

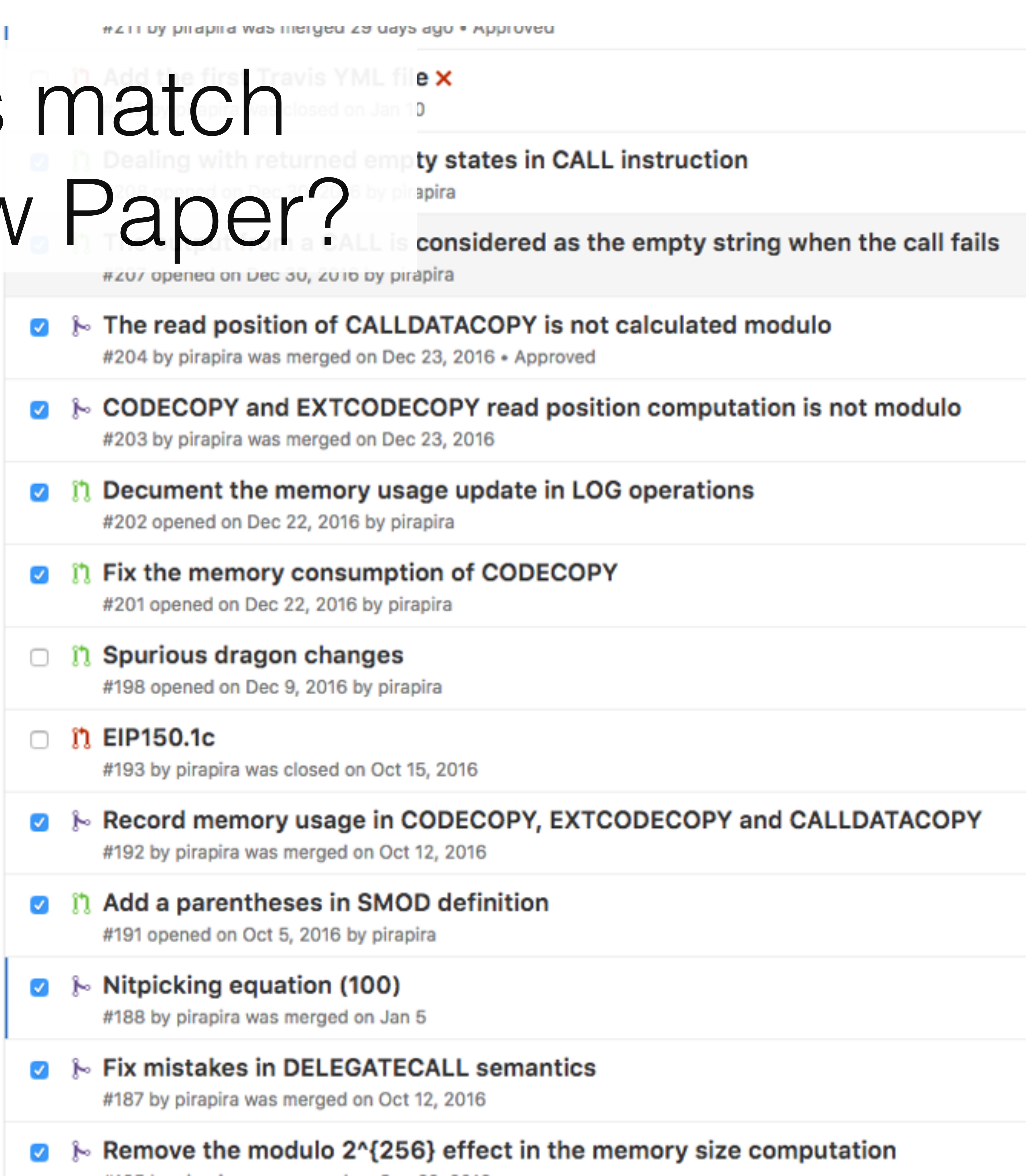
Works the same as Ethereum Virtual Machine in C++ etc.

Does this match the Yellow Paper?

- **No**.
  See pull-requests

- If you can review LaTeX, that's great help!

  Changes coming.



#211 by pirapira was merged 29 days ago • Approved

Add the first Travis YML file ×
by pirapira was closed on Jan 10

Dealing with returned empty states in CALL instruction
#206 opened on Dec 30, 2016 by pirapira

The output from a CALL is considered as the empty string when the call fails
#207 opened on Dec 30, 2016 by pirapira

The read position of CALLDATACOPY is not calculated modulo
#204 by pirapira was merged on Dec 23, 2016 • Approved

CODECOPY and EXTCODECOPY read position computation is not modulo
#203 by pirapira was merged on Dec 23, 2016

Decument the memory usage update in LOG operations
#202 opened on Dec 22, 2016 by pirapira

Fix the memory consumption of CODECOPY
#201 opened on Dec 22, 2016 by pirapira

Spurious dragon changes
#198 opened on Dec 9, 2016 by pirapira

EIP150.1c
#193 by pirapira was closed on Oct 15, 2016

Record memory usage in CODECOPY, EXTCODECOPY and CALLDATACOPY
#192 by pirapira was merged on Oct 12, 2016

Add a parentheses in SMOD definition
#191 opened on Oct 5, 2016 by pirapira

Nitpicking equation (100)
#188 by pirapira was merged on Jan 5

Fix mistakes in DELEGATECALL semantics
#187 by pirapira was merged on Oct 12, 2016

Remove the modulo 2^{256} effect in the memory size computation

Small Community

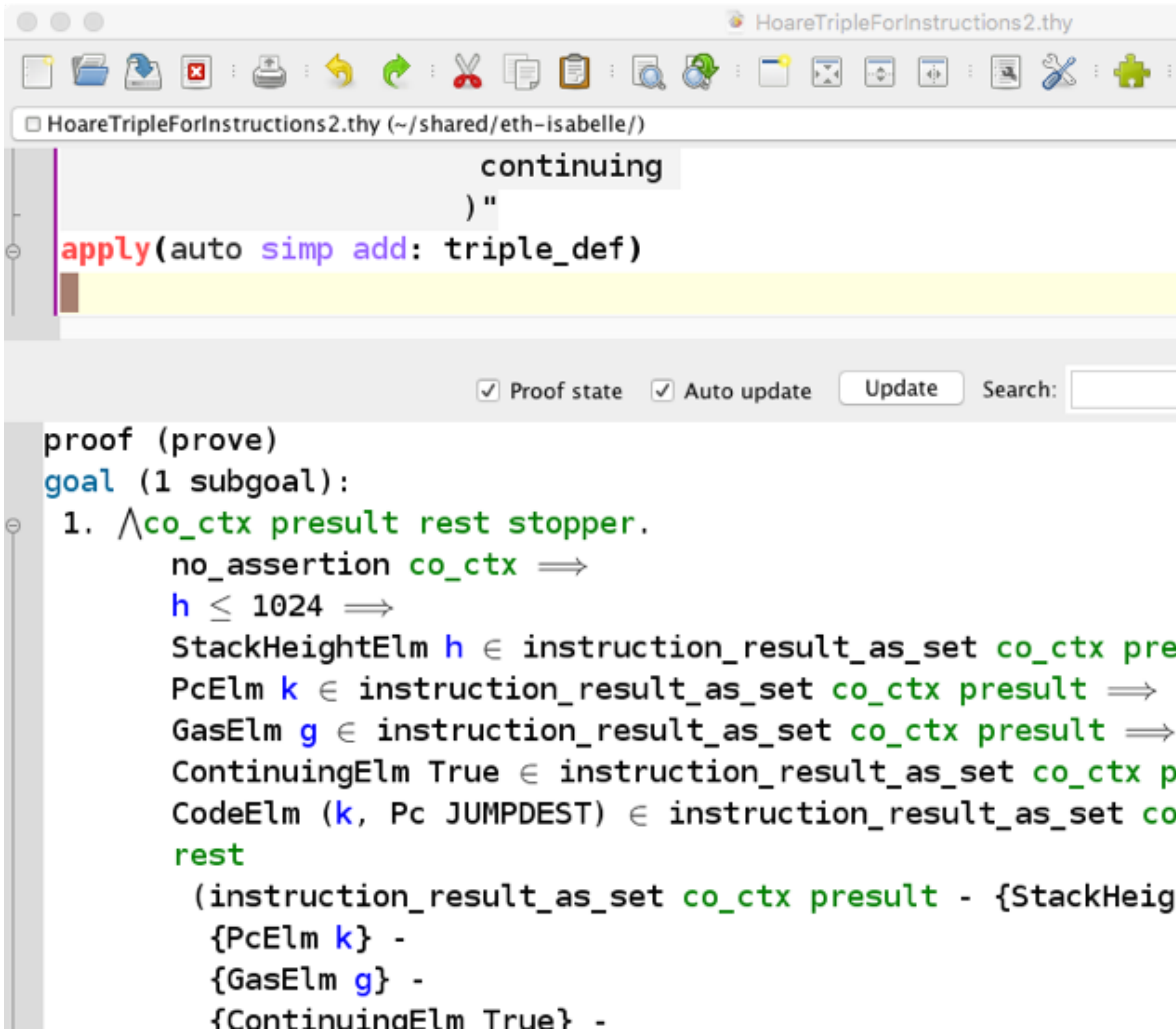
- EVM definition available for Isabelle/HOL and HOL4 (Coq is coming, thanks to somebody..)
- received some external contributions
- some researchers started projects

Proof IDE

input commands

current goal shown

you can jump
to definitions etc.

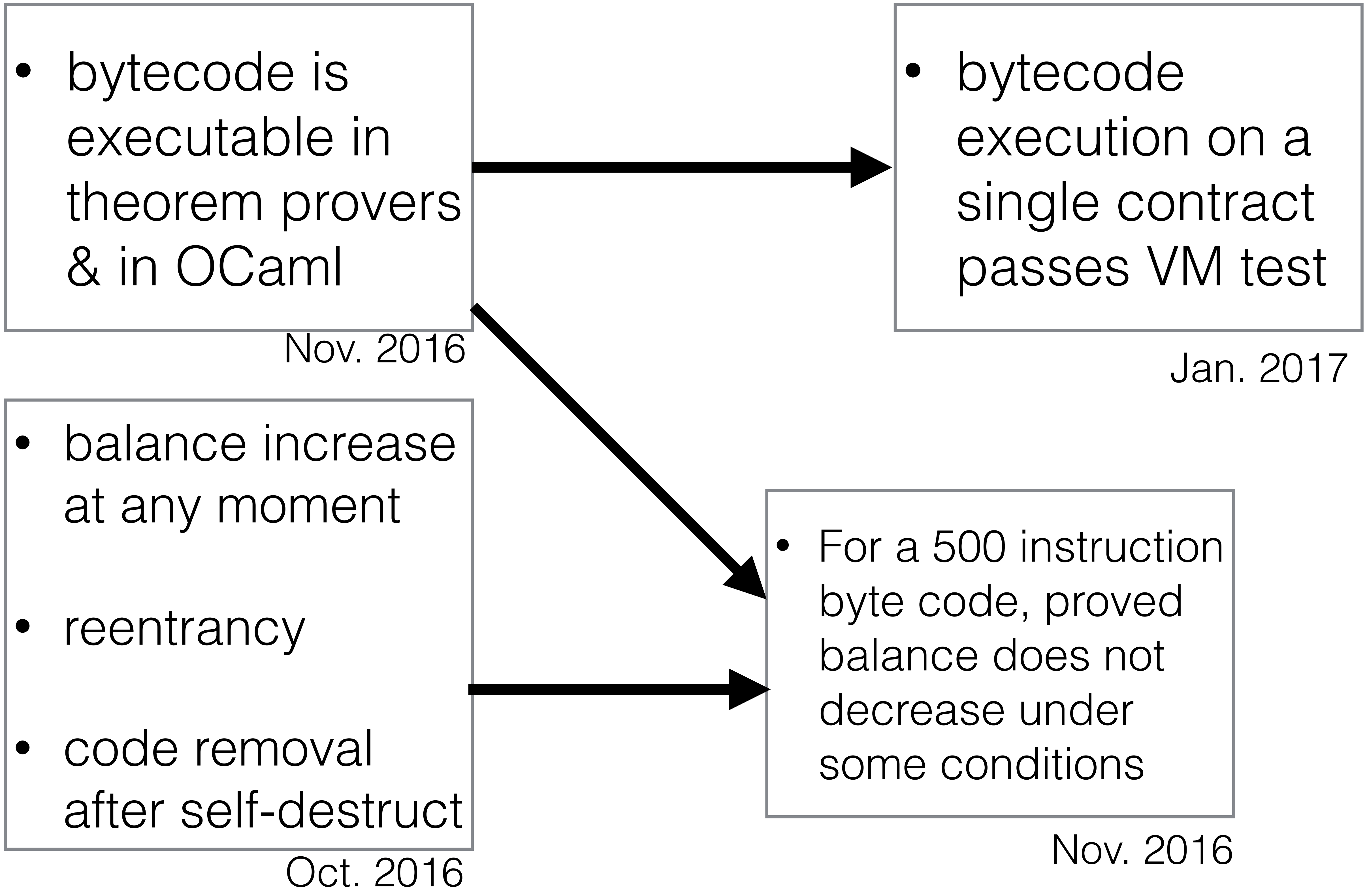


HoareTripleForInstructions2.thy
HoareTripleForInstructions2.thy (~/shared/eth-isabelle/)
Proof state
Auto update
Update
Search:

```
                    continuing
                   )"
apply(auto simp add: triple_def)
```

```
proof (prove)
goal (1 subgoal):
 1. ⋀co_ctx presult rest stopper.
       no_assertion co_ctx ⟹
       h ≤ 1024 ⟹
       StackHeightElm h ∈ instruction_result_as_set co_ctx pre
       PcElm k ∈ instruction_result_as_set co_ctx presult ⟹
       GasElm g ∈ instruction_result_as_set co_ctx presult ⟹
       ContinuingElm True ∈ instruction_result_as_set co_ctx p
       CodeElm (k, Pc JUMPDEST) ∈ instruction_result_as_set co
       rest
        (instruction_result_as_set co_ctx presult - {StackHeig
         {PcElm k} -
         {GasElm g} -
         {ContinuingElm True} -
```

So far: EVM definitions ready



- bytecode is executable in theorem provers & in OCaml

Nov. 2016

- bytecode execution on a single contract passes VM test

Jan. 2017

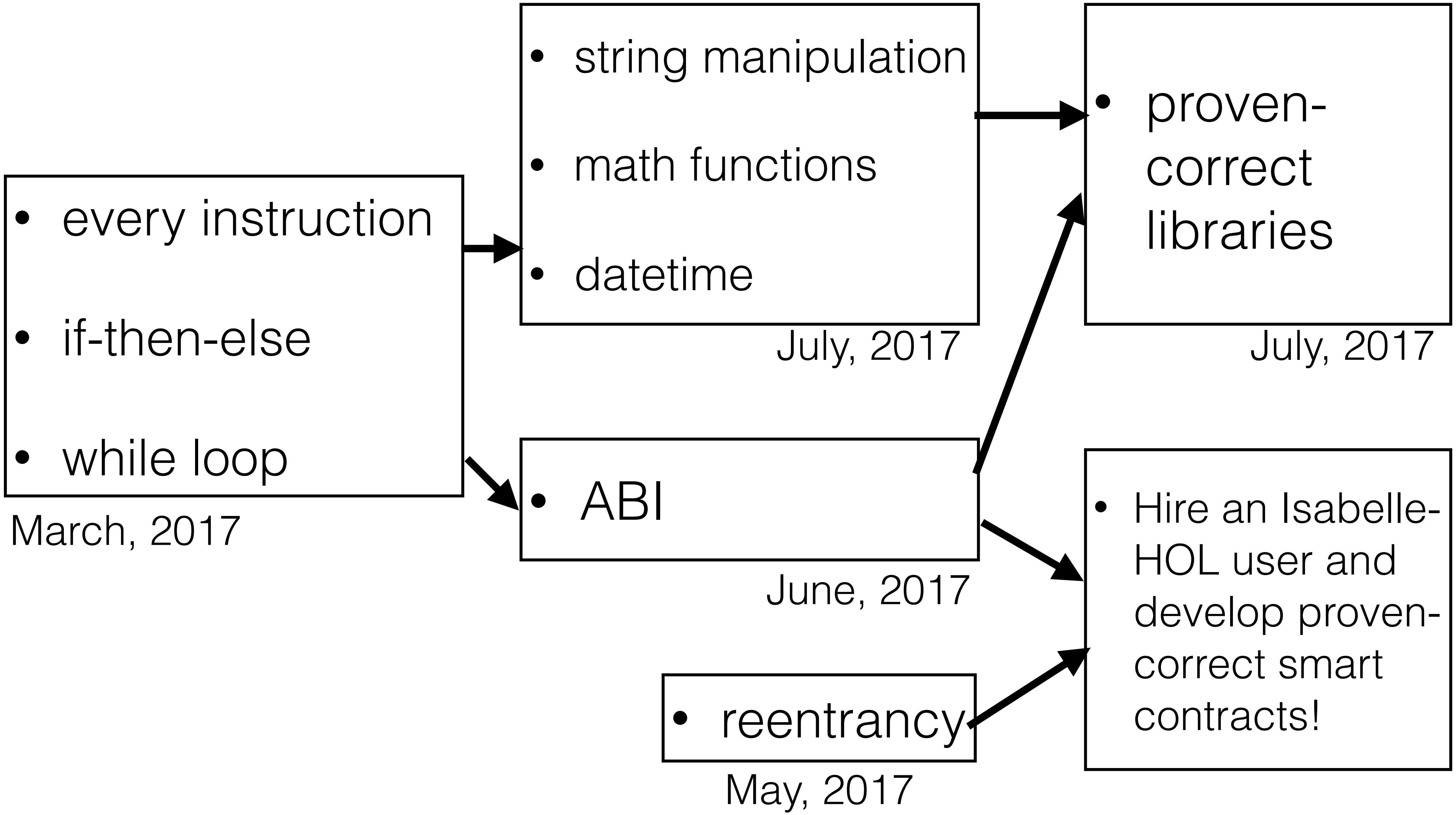
- balance increase at any moment
- reentrancy
- code removal after self-destruct

Oct. 2016

- For a 500 instruction byte code, proved balance does not decrease under some conditions

Nov. 2016

Plan: reusable verified snippets



Don’t trust it

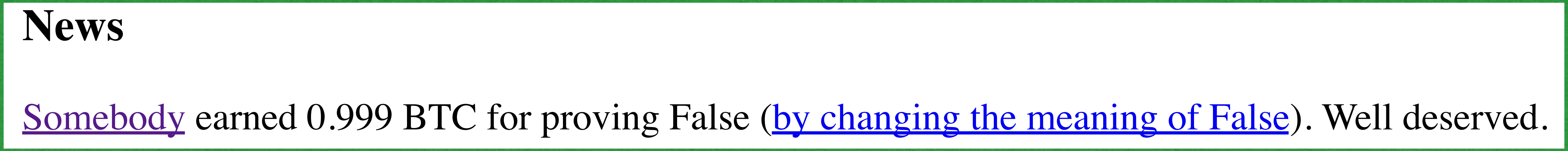
- because, just see what happened to the proof market

(obsolete) bug bounty program: prove falsehood and get a bitcoin

```
Theorem f : False.
Proof.
  (* fill in and change Admitted into Qed *)
Admitted.
```

I put 0.999 BTC as bounty.

2014



News

Somebody earned 0.999 BTC for proving False (by changing the meaning of False). Well deserved.

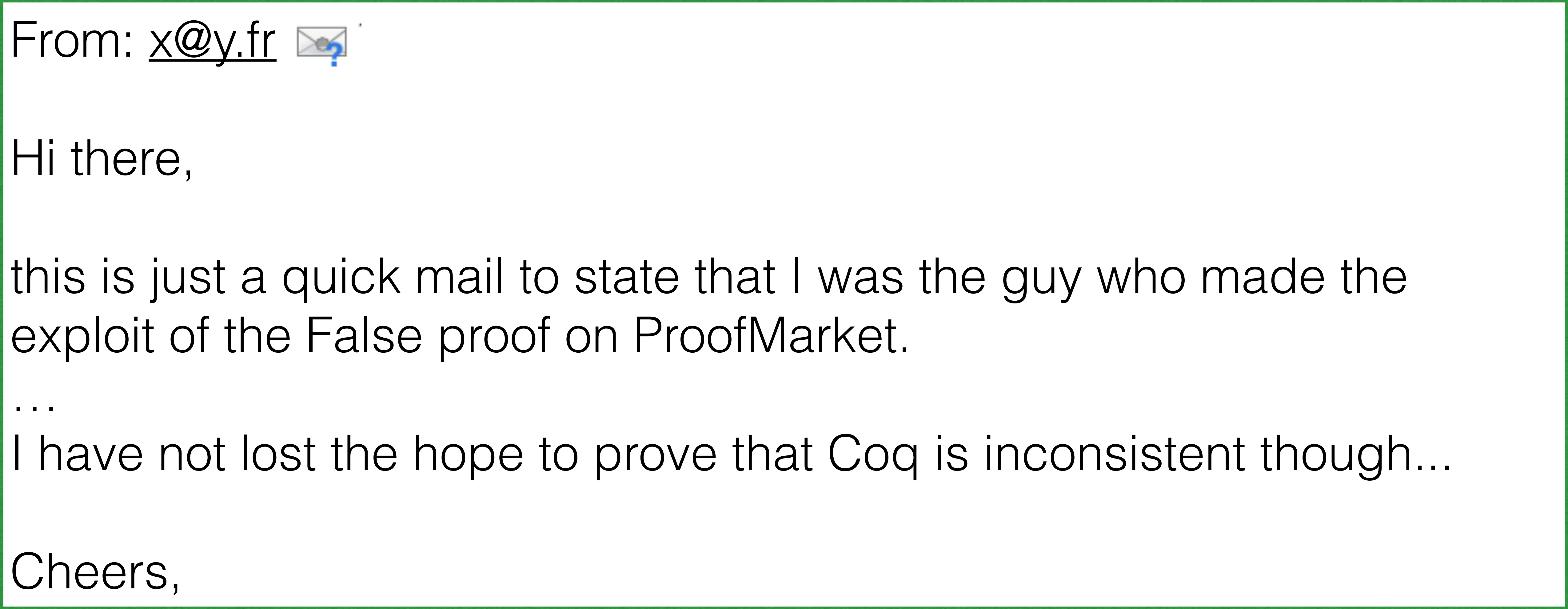
2014

The proof of False that I bought with 1 BTC

```
Inductive False := I.

Theorem inhabitant : False.
Proof.
exact I.
Qed.
```

2014


From: x@y.fr

Hi there,

this is just a quick mail to state that I was the guy who made the exploit of the False proof on ProofMarket.

…

I have not lost the hope to prove that Coq is inconsistent though...

Cheers,

Indeed, anything was provable in Coq.
I closed the site.

I trust bounties more than proof checkers.

2014

Another bounty (what could go wrong)

- I proved a wallet correct (to a spec). I put 1,000 ETH at 0x0fcc015903e7e51a947ed7276a21d37a11b29e61
- Please try to take the fund
- The first one is as simple as “prove False”
- A blog post is scheduled 1pm today on medium.
- (From here, I’ll set up more and more complicated proven-correct Ethereum contracts.)

Projects waiting for you

- Resource Consumption by Gas
  There is a proof on github:
  “With G gas, only G steps are possible”.
  # of (CALL, EXTCODE, BALANCE, SSTOREs)
- CompCert-style proven-correct compiler into EVM
- Proven-correct transpiler from EVM to eWASM
- CSmith-style Solidity compiler fuzzing

@pirapira (twitter, GitHub)

blockchains

theorem proving

strong internal consistency

by deterministic rules & cryptographic hashes

by small number of trusted rules & contradiction explodes

limited external interface

because distributed nodes see the world differently

because it can only talk about mathematics defined within

```
lemma whole_program_invalid_caller:
"triple {OutOfGas} (⟨unat bn ≥ 2463000 ∧ ucast c ≠ w⟩ **
            block_number_pred bn **
            stack_height 0 **
            program_counter 0 ** caller c **
            storage (word_rcat [0]) w **
            gas_pred g **
            continuing
           )
           whole_concrete_program
           (block_number_pred bn **
            stack_height 0 **
            program_counter 8 ** caller c **
            storage (word_rcat [0]) w **
            gas_pred
               (g + (- Gsload (unat bn) - 2)
                - 2 * Gverylow - Gverylow - Ghigh) **
            not_continuing ** action (ContractReturn []))"
```

```
lemma check_pass_whole_concrete:
  "triple {OutOfGas} (⟨unat bn ≥ 2463000 ⟩ **
                      block_number_pred bn **
                      stack_height 0 **
                      program_counter 0 ** caller c **
                      storage (word_rcat [0]) (ucast c) **
                      gas_pred g **
                      continuing **
                      this_account t **
                      balance t b **
                      memory_usage 0
                     )
                     whole_concrete_program
                     (memory_usage 0 **
                      stack_topmost 0 [] **
                      program_counter 22 **
                      this_account t **
                      balance t 0 **
                      gas_any **
                      not_continuing **
                      action (ContractCall ⦇
                        callarg_gas = word_rcat [(8 :: byte), 0]
                      , callarg_code = c
                      , callarg_recipient = c
                      , callarg_value = b
                      , callarg_data = []
                      , callarg_output_begin = word_rcat [0]
                      , callarg_output_size = word_rcat [0] ⦈) **
                      block_number_pred bn **
                      caller c **
                      storage (word_rcat [0]) (ucast c)
                     )"
```