formal verification of smart contracts

Ethereum Meetup 1.11.2016, Berlin

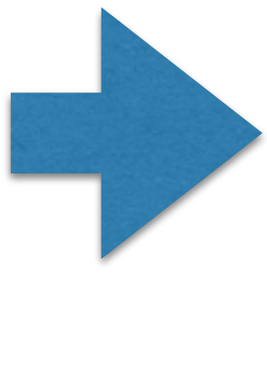
Yoichi Hirai

“formal” verification?

http://www.telegraph.co.uk/culture/tvandradio/10803323/Why-cant-we-make-drama-like-The-Pallisers-anymore.html

“formal” comes from formalised math

Usual Math

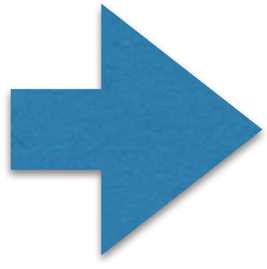


idea

correct but boring

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

Formalised Math



no idea

just looking at “form”

correct

formal proofs on twitter

Messages

Search Twitter

@ipc_bot (A ∧ B) -> B

GIF

118

Tweet

IPC bot

@ipc_bot

.@pirapira Provable. (pwl

$A \wedge B \to B$:
Provable.
Proof tree (intuitionistic):

$$\dfrac{\dfrac{[A \wedge B]_1}{B}\,\wedge E_2}{A \wedge B \to B} \to I(1$$

8:00 PM - 31 Oct 2016

German tech from 1934

(*N*)-Zeichen-„Einführung“ (*E*) bzw. „-Beseitigung“ (*B*). in § 5.

Die Schlußfiguren-Schemata:

$$\mathfrak{U}\ E \qquad \mathfrak{U}\ B \qquad \mathfrak{O}\ E$$

$$\frac{\mathfrak{A} \quad \mathfrak{B}}{\mathfrak{A} \,\&\, \mathfrak{B}} \qquad \frac{\mathfrak{A} \,\&\, \mathfrak{B}}{\mathfrak{A}} \qquad \frac{\mathfrak{A} \,\&\, \mathfrak{B}}{\mathfrak{B}} \qquad \frac{\mathfrak{A}}{\mathfrak{A} \vee \mathfrak{B}} \qquad \frac{\mathfrak{B}}{\mathfrak{A} \vee \mathfrak{B}}$$

Und-Einführung

Und-Beseitigung

Oder-Einführung

All mathematical logicians are encouraged to learn German especially if they become an Ethereum DΞV in Berlin.

Good rules and bad rules

Good rules allow no smuggling.

Bad rules allow smuggling.

Und-Einführung

Und-Beseitigung

$$\frac{\mathfrak{A} \quad \mathfrak{B}}{\mathfrak{A} \,\&\, \mathfrak{B}} \qquad \frac{\mathfrak{A} \,\&\, \mathfrak{B}}{\mathfrak{B}}$$

$$\frac{A}{A \ \mathit{tonk}\ B} \qquad \frac{A \ \mathit{tonk}\ B}{B}$$

tonk-introduction

tonk-elimination

$$\frac{\dfrac{\text{Truth}}{\text{Truth } \mathit{tonk} \text{ Falsehood}}}{\text{Falsehood}}$$

Belnap: *Tonk, Plonk and Plink* (1962).

What are the inference rules for Solidity programs?

Solidity code

idea

this works

but why do you believe the code executes top to bottom, not bottom to up?

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

Solidity code

knows the rules, can compile

correct

EVM bytecode

What are the inference rules for EVM programs?

PUSH1 0x60
PUSH1 0x40
MSTORE
…

stack: [0x60]
memory: empty

does not go wrong

PUSH1 0x60
PUSH1 0x40
MSTORE
…

stack: empty
memory: empty

does not go wrong

The hierarchy of programming languages

- spec authoritative written for proof checkers: CakeML

- spec translated into proof checkers: C11, javascript (ECMAScript 5)

- spec with mathematical definitions: Standard ML, Scheme, (EVM?)

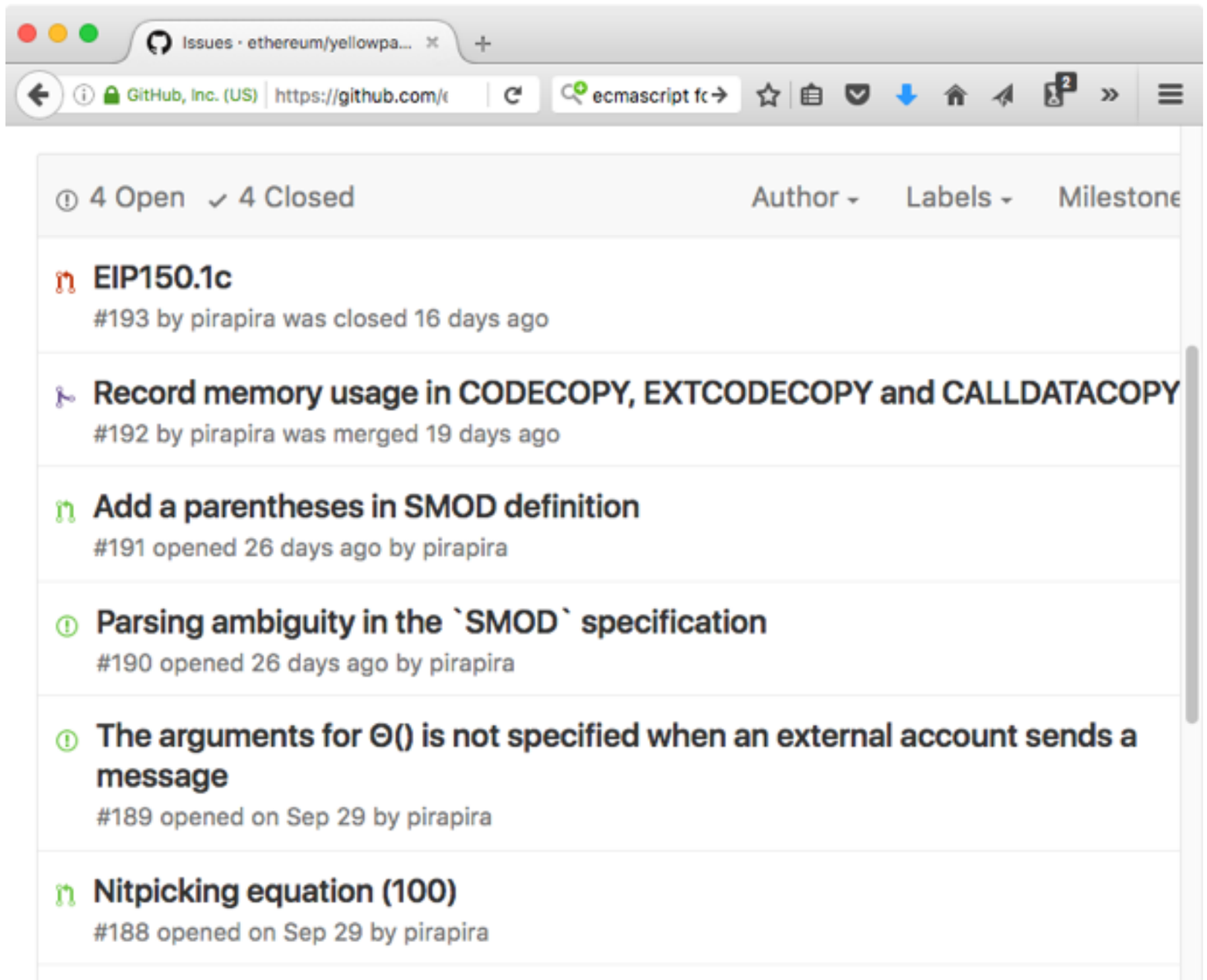
- spec for people: EVM, C++14, Haskell2010, Ruby

- no spec: Solidity

https://commons.wikimedia.org/wiki/File:Francesco_Botticini_-_The_Assumption_of_the_Virgin.jpg

EVM spec in the yellow paper

- There are bugs, but the spec exists, and different implementations.



Type every rule in a proof checker

- just implementing another EVM
- around 50 pages in A4
- the proof checker Isabelle/HOL does not allow "tonk"s

POP removes the topmost element of the stack

abbreviation *pop* ::
variable-env ⇒ *constant-env* ⇒ *instruction-result*
where
pop v c ≡ *InstructionContinue* (*venv-advance-pc c*
v(|*venv-stack* := *tl* (*venv-stack v*)|))

~~Typing~~ pasting your program in a proof checker

definition *deed-insts* :: *inst list*
where *deed-insts* =
Stack (*PUSH-N* [*0x60*]) #
Stack (*PUSH-N* [*0x40*]) #
Memory MSTORE #
Info CALLDATASIZE #
Arith ISZERO #
Stack (*PUSH-N* [*0x00*, *0x6c*]) #
Pc JUMPI #
Stack (*PUSH-N* [*0xe0*]) #
Stack (*PUSH-N* [*0x02*]) #
Arith EXP #
Stack (*PUSH-N* [*0x00*]) #
Stack CALLDATALOAD #
Arith DIV #
Stack (*PUSH-N* [*0x05*, *0xb3*, *0x44*, *0x10*]) #
Dup 2 #
Arith inst-EQ #
Stack (*PUSH-N* [*0x00*, *0x6e*]) #
Pc JUMPI #
Stack (*PUSH-N* [*0x0b*, *0x5a*, *0xb3*, *0xd5*]) #
Arith inst-EQ #
Stack (*PUSH-N* [*0x00*, *0x7c*]) #
Pc JUMPI #
Dup 1 #
Stack (*PUSH-N* [*0x13*, *0xaf*, *0x40*, *0x35*]) #
Arith inst-EQ #
Stack (*PUSH-N* [*0x00*, *0x89*]) #
Pc JUMPI #
Dup 1 #
Stack (*PUSH-N* [*0x2b*, *0x20*, *0xe3*, *0x97*]) #
Arith inst-EQ #
Stack (*PUSH-N* [*0x00*, *0xaf*]) #
Pc JUMPI #
Dup 1 #
Stack (*PUSH-N* [*0x8d*, *0xa5*, *0xcb*, *0x5b*]) #
Arith inst-EQ #
Stack (*PUSH-N* [*0x00*, *0xc6*]) #
Pc JUMPI #
Dup 1 #
Stack (*PUSH-N* [*0xbb*, *0xe4*, *0x27*, *0x71*]) #
Arith inst-EQ #
Stack (*PUSH-N* [*0x00*, *0xdd*]) #
Pc JUMPI #
Dup 1 #
Stack (*PUSH-N* [*0xfa*, *0xab*, *0x9d*, *0x39*]) #
Arith inst-EQ #
Stack (*PUSH-N* [*0x01*, *0x03*]) #
Pc JUMPI #
Dup 1 #
Stack (*PUSH-N* [*0xfb*, *0x16*, *0x69*, *0xca*]) #
Arith inst-EQ #
Stack (*PUSH-N* [*0x01*, *0x29*]) #
Pc JUMPI #
Pc JUMPDEST #
Misc STOP #
Pc JUMPDEST #

Info CALLVALUE #
Stack (*PUSH-N* [*0x00*, *0x02*]) #
Pc JUMPI #
Stack (*PUSH-N* [*0x01*, *0x4a*]) #
Stack (*PUSH-N* [*0x01*]) #
Storage SLOAD #
Dup 2 #
Pc JUMP #
Pc JUMPDEST #
Info CALLVALUE #
Stack (*PUSH-N* [*0x00*, *0x02*]) #
Pc JUMPI #
Stack (*PUSH-N* [*0x00*, *0x6c*]) #
Stack (*PUSH-N* [*0x01*, *0x89*]) #
Pc JUMP #
Pc JUMPDEST #
Info CALLVALUE #
Stack (*PUSH-N* [*0x00*, *0x02*]) #
Pc JUMPI #
Stack (*PUSH-N* [*0x00*, *0x6c*]) #
Stack (*PUSH-N* [*0x04*]) #
Stack CALLDATALOAD #
Stack (*PUSH-N* [*0x00*]) #
Storage SLOAD #
Info CALLER #
Stack (*PUSH-N* [*0x01*]) #
Stack (*PUSH-N* [*0xa0*]) #
Stack (*PUSH-N* [*0x02*]) #
Arith EXP #
Arith SUB #
Swap 1 #
Dup 2 #
Bits inst-AND #
Swap 2 #
Bits inst-AND #
Arith inst-EQ #
Stack (*PUSH-N* [*0xf8*, *0xff*]) #
Pc JUMPI #
Stack (*PUSH-N* [*0x00*, *0x02*]) #
Pc JUMP #
Pc JUMPDEST #
Info CALLVALUE #
Stack (*PUSH-N* [*0x00*, *0x02*]) #
Pc JUMPI #
Stack (*PUSH-N* [*0x00*, *0x6c*]) #
Stack (*PUSH-N* [*0x04*]) #
Stack CALLDATALOAD #
Stack (*PUSH-N* [*0x00*]) #
Storage SLOAD #
Info CALLER #
Stack (*PUSH-N* [*0x01*]) #
Stack (*PUSH-N* [*0xa0*]) #
Stack (*PUSH-N* [*0x02*]) #
Arith EXP #
Arith SUB #
Swap 1 #
Dup 2 #
Bits inst-AND #
Swap 2 #
Bits inst-AND #
Arith inst-EQ #
Stack (*PUSH-N* [*0x02*, *0xc7*]) #
Pc JUMPI #
Stack (*PUSH-N* [*0x00*, *0x02*]) #
Pc JUMP #
Pc JUMPDEST #
Stack (*PUSH-N* [*0x00*, *0x6c*]) #
Stack (*PUSH-N* [*0x04*]) #
Stack CALLDATALOAD #
Stack (*PUSH-N* [*0x00*]) #
Storage SLOAD #
Info CALLER #
Stack (*PUSH-N* [*0x01*]) #
Stack (*PUSH-N* [*0xa0*]) #
Stack (*PUSH-N* [*0x02*]) #
Arith EXP #
Arith SUB #
Swap 1 #
Dup 2 #
Bits inst-AND #
Swap 2 #
Bits inst-AND #
Arith inst-EQ #
Stack (*PUSH-N* [*0x02*, *0xe9*]) #
Pc JUMPI #
Stack (*PUSH-N* [*0x00*, *0x02*]) #
Pc JUMP #
Pc JUMPDEST #
Stack (*PUSH-N* [*0x40*]) #
Dup 1 #
Memory MLOAD #
Swap 2 #
Dup 3 #
Memory MSTORE #
Memory MLOAD #
Swap 1 #
Dup 2 #
Swap 1 #
Arith SUB #
Stack (*PUSH-N* [*0x20*]) #
Arith ADD #
Swap 1 #
Misc RETURN #
Pc JUMPDEST #
Stack (*PUSH-N* [*0x40*]) #
Memory MLOAD #
Stack (*PUSH-N* [*0x02*]) #
Stack (*PUSH-N* [*0x00*, *0x02*]) #
Pc JUMP #
Pc JUMPDEST #
Info CALLVALUE #
Stack (*PUSH-N* [*0x00*, *0x02*]) #
Pc JUMPI #
Stack (*PUSH-N* [*0x01*, *0xa0*]) #
Stack (*PUSH-N* [*0x00*]) #
Storage SLOAD #
Stack (*PUSH-N* [*0x01*]) #
Stack (*PUSH-N* [*0xa0*]) #

Pc JUMPI #
Stack (*PUSH-N* [*0x00*, *0x6c*]) #
Stack (*PUSH-N* [*0x04*]) #
Stack CALLDATALOAD #
Stack (*PUSH-N* [*0x00*]) #
Storage SLOAD #
Info CALLER #
Stack (*PUSH-N* [*0x01*]) #
Stack (*PUSH-N* [*0xa0*]) #
Stack (*PUSH-N* [*0x02*]) #
Arith EXP #
Arith SUB #
Swap 1 #
Dup 2 #
Bits inst-AND #
Swap 2 #
Bits inst-AND #
Arith inst-EQ #
Stack (*PUSH-N* [*0x02*, *0xc7*]) #
Pc JUMPI #
Stack (*PUSH-N* [*0x00*, *0x02*]) #
Pc JUMP #
Pc JUMPDEST #
Stack (*PUSH-N* [*0x00*, *0x6c*]) #
Stack (*PUSH-N* [*0x04*]) #
Stack CALLDATALOAD #
Stack (*PUSH-N* [*0x00*]) #
Storage SLOAD #
Info CALLER #
Stack (*PUSH-N* [*0x01*]) #
Stack (*PUSH-N* [*0xa0*]) #
Stack (*PUSH-N* [*0x02*]) #
Arith EXP #
Arith SUB #
Swap 1 #
Dup 2 #
Bits inst-AND #
Swap 2 #
Bits inst-AND #
Arith inst-EQ #
Stack (*PUSH-N* [*0x02*, *0xe9*]) #
Pc JUMPI #
Stack (*PUSH-N* [*0x00*, *0x02*]) #
Pc JUMP #
Pc JUMPDEST #
Dup 1 #
Memory MLOAD #
Swap 2 #
Dup 3 #
Memory MSTORE #
Memory MLOAD #
Swap 1 #
Dup 2 #
Swap 1 #
Arith SUB #
Stack (*PUSH-N* [*0x20*]) #
Arith ADD #
Swap 1 #
Misc RETURN #
Pc JUMPDEST #
Stack (*PUSH-N* [*0x40*]) #
Memory MLOAD #
Stack (*PUSH-N* [*0x02*]) #
Storage SLOAD #
Stack (*PUSH-N* [*0x01*]) #
Stack (*PUSH-N* [*0xa0*]) #
Stack (*PUSH-N* [*0x02*]) #
Arith EXP #
Arith SUB #
Swap 1 #
Dup 2 #
Bits inst-AND #
Swap 2 #
Info ADDRESS #
Swap 1 #
Swap 2 #
Bits inst-AND #
Info BALANCE #
Dup 1 #
Arith ISZERO #
Stack (*PUSH-N* [*0x08*, *0xfc*]) #
Arith MUL #
Swap 2 #
Arith inst-AND #
Swap 1 #
Swap 2 #
Bits inst-AND #
Info BALANCE #
Dup 1 #
Arith ISZERO #
Stack (*PUSH-N* [*0x08*, *0xfc*]) #
Arith MUL #
Swap 2 #
Arith inst-AND #
Swap 1 #
Swap 2 #
Bits inst-AND #
Info BALANCE #
Dup 1 #
Arith ISZERO #
Stack (*PUSH-N* [*0x01*, *0xf5*]) #
Pc JUMPI #
Stack (*PUSH-N* [*0xde*, *0xad*]) #
Misc SUICIDE #
Pc JUMPDEST #
Stack (*PUSH-N* [*0x00*, *0x02*]) #
Pc JUMP #
Pc JUMPDEST #
Stack (*PUSH-N* [*0x02*]) #
Dup 1 #
Storage SLOAD #
Stack (*PUSH-N* [*0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*]) #
Bits inst-NOT #
Bits inst-AND #
Dup 3 #
Bits inst-OR #
Swap 1 #
Storage SSTORE #
Stack (*PUSH-N* [*0x40*]) #
Memory MLOAD #
Stack (*PUSH-N* [*0xde*, *0xad*]) #
Swap 1 #
Stack (*PUSH-N* [*0x03*, *0xe8*]) #
Info ADDRESS #
Stack (*PUSH-N* [*0x01*]) #
Stack (*PUSH-N* [*0xa0*]) #
Stack (*PUSH-N* [*0x02*]) #
Arith EXP #
Arith SUB #
Bits inst-AND #
Info BALANCE #
Dup 5 #
Dup 3 #
Arith SUB #
Arith MUL #
Swap 2 #
Stack (*PUSH-N* [*0x00*]) #

Dup 3 #
Memory MSTORE #
Memory MLOAD #
Swap 1 #
Dup 2 #
Swap 1 #
Arith SUB #
Stack (*PUSH-N* [*0x20*]) #
Arith ADD #
Swap 1 #
Misc RETURN #
Pc JUMPDEST #
Stack (*PUSH-N* [*0x40*]) #
Memory MLOAD #
Stack (*PUSH-N* [*0x02*]) #
Storage SLOAD #
Stack (*PUSH-N* [*0x01*]) #
Stack (*PUSH-N* [*0xa0*]) #
Stack (*PUSH-N* [*0x02*]) #
Arith EXP #
Arith SUB #
Swap 1 #
Dup 2 #
Bits inst-AND #
Swap 2 #
Info ADDRESS #
Swap 1 #
Swap 2 #
Bits inst-AND #
Info BALANCE #
Dup 1 #
Arith ISZERO #
Stack (*PUSH-N* [*0x08*, *0xfc*]) #
Arith MUL #
Swap 2 #
Arith inst-AND #
Swap 1 #
Swap 2 #
Bits inst-AND #
Info BALANCE #
Dup 1 #
Arith ISZERO #
Stack (*PUSH-N* [*0x01*, *0xf5*]) #
Pc JUMPI #
Stack (*PUSH-N* [*0xde*, *0xad*]) #
Misc SUICIDE #
Pc JUMPDEST #
Stack (*PUSH-N* [*0x00*, *0x02*]) #
Pc JUMP #
Pc JUMPDEST #
Stack (*PUSH-N* [*0x02*]) #
Dup 1 #
Storage SLOAD #
Stack (*PUSH-N* [*0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*]) #
Bits inst-NOT #
Bits inst-AND #
Dup 3 #
Bits inst-OR #
Swap 1 #
Storage SSTORE #
Stack (*PUSH-N* [*0x40*]) #
Memory MLOAD #
Stack (*PUSH-N* [*0xde*, *0xad*]) #
Swap 1 #
Stack (*PUSH-N* [*0x03*, *0xe8*]) #
Info ADDRESS #
Stack (*PUSH-N* [*0x01*]) #
Stack (*PUSH-N* [*0xa0*]) #
Stack (*PUSH-N* [*0x02*]) #
Arith EXP #
Arith SUB #
Bits inst-AND #
Info BALANCE #
Dup 5 #
Dup 3 #
Arith SUB #
Arith MUL #
Swap 2 #
Stack (*PUSH-N* [*0x00*]) #

Stack (*PUSH-N* [*0x02*]) #
Storage SLOAD #
Stack (*PUSH-N* [*0xa0*]) #
Stack (*PUSH-N* [*0x02*]) #
Arith EXP #
Swap 1 #
Arith DIV #
Stack (*PUSH-N* [*0xff*]) #
Bits inst-AND #
Arith ISZERO #
Stack (*PUSH-N* [*0x02*, *0x6f*]) #
Pc JUMPI #
Stack (*PUSH-N* [*0x00*, *0x02*]) #
Pc JUMP #
Pc JUMPDEST #
Stack (*PUSH-N* [*0x02*]) #
Dup 1 #
Storage SLOAD #
Stack (*PUSH-N* [*0xff*, *0x00*, *0x00*, *0x00*, *0x00*, *0x00*, *0x00*, *0x00*, *0x00*, *0x00*, *0x00*, *0x00*, *0x00*, *0x00*, *0x00*, *0x00*]) #
Bits inst-NOT #
Bits inst-AND #
Swap 1 #
Storage SSTORE #
Stack (*PUSH-N* [*0x40*]) #
Memory MLOAD #
Stack (*PUSH-N* [*0xde*, *0xad*]) #
Swap 1 #
Stack (*PUSH-N* [*0x03*, *0xe8*]) #
Info ADDRESS #
Stack (*PUSH-N* [*0x01*]) #
Stack (*PUSH-N* [*0xa0*]) #
Stack (*PUSH-N* [*0x02*]) #
Arith EXP #
Arith SUB #
Bits inst-AND #
Info BALANCE #
Dup 5 #
Dup 3 #
Arith SUB #
Arith MUL #
Swap 2 #
Stack (*PUSH-N* [*0x00*]) #

Stack POP #
Stack POP #
Stack POP #
Stack POP #
Arith ISZERO #
Arith ISZERO #
Stack (*PUSH-N* [*0x02*, *0x54*]) #
Pc JUMPI #
Stack (*PUSH-N* [*0x00*, *0x02*]) #
Pc JUMP #
□

Dup 2 #
Dup 2 #
Dup 2 #
Dup 6 #
Dup 9 #
Dup 9 #
Misc CALL #
Swap 4 #
Stack POP #
Stack POP #
Stack POP #
Stack POP #
Arith ISZERO #
Arith ISZERO #
Stack (*PUSH-N* [*0x01*, *0x5c*]) #
Pc JUMPI #
Stack (*PUSH-N* [*0x00*, *0x02*]) #
Pc JUMP #
Pc JUMPDEST #
Stack (*PUSH-N* [*0x00*]) #
Dup 1 #
Storage SLOAD #
Stack (*PUSH-N* [*0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*, *0xff*]) #
Bits inst-NOT #
Bits inst-AND #
Dup 3 #
Bits inst-OR #
Swap 1 #
Storage SSTORE #
Stack (*PUSH-N* [*0x01*]) #
Stack (*PUSH-N* [*0xa0*]) #
Stack (*PUSH-N* [*0x02*]) #
Arith EXP #
Arith SUB #
Swap 2 #
Dup 3 #
Bits inst-AND #
Swap 2 #
Info ADDRESS #
Bits inst-AND #
Info BALANCE #
Dup 1 #
Swap 1 #
Arith SUB #
Dup 1 #
Arith ISZERO #
Stack (*PUSH-N* [*0x08*, *0xfc*]) #
Arith MUL #
Swap 2 #
Stack (*PUSH-N* [*0x00*]) #
Dup 2 #
Dup 2 #
Dup 2 #
Dup 6 #
Dup 9 #
Dup 9 #
Misc CALL #
Swap 4 #

Final question:
what do you want?

- The account should do expected things.
  I have to interview people. Not right now.

- The account should not do anything wrong.

Final question:
what do you want?

- The account should not do anything wrong.

  The balance should not decrease unless an authorised account tells so.

Final question: what do you want?

- The account should not do anything wrong.

  The balance should not decrease unless an authorised account tells so.

  A non-authorised account cannot authorise any account.

Typing your expectations in a proof checker

under these conditions

certain bad things never happen

$$
\begin{array}{l}
(* \ \textit{the caller is not the regsitrar}; \ *) \\
\quad \textit{ucast} \ (\textit{account-storage init-state 0}) \neq \textit{callenv-caller init-call} \\
\\
\quad (* \ \textit{the Deed contract is still marked active}; \ *) \\
\wedge \ (\textit{255 AND account-storage init-state 2 div 2 ^ 160} \neq 0) \\
\\
\quad (* \ \textit{the call does not overflow the balance}; \ *) \\
\wedge \ \textit{account-balance init-state} + \textit{callenv-value init-call} \geq \textit{account-balance init-state} \\
\\
\quad (* \ \textit{the account is not marked as destroyed.} \ *) \\
\wedge \ \neg \ (\textit{account-killed init-state})
\end{array}
$$

$$
\begin{array}{l}
\quad (* \ \textit{The balance has not decreased.} \ *) \\
\quad \textit{account-balance init-state} \leq \textit{account-balance post-state} \\
\\
\quad (* \ \textit{The account is still not marked for destruction} \\
\qquad (\textit{i.e. the account has not executed self-destruction}). \ *) \\
\wedge \ \neg \ (\textit{account-killed post-state}) \\
\\
\quad (* \ \textit{The Deed contract is still marked as active.} \ *) \\
\wedge \ (\textit{255 AND account-storage post-state 2 div 2 ^ 160} \neq 0) \\
\\
\quad (* \ \textit{The registrar of the contract remains the same.} \ *) \\
\wedge \ \textit{account-storage init-state 0} = \textit{account-storage post-state 0})
\end{array}
$$

prove!
if you've forgotten something…

Isabelle2016 - Deed.thy

File Edit Search Markers Folding View Utilities Macros Plugins Help

Deed.thy (~/src/eth-isabelle/example/)

```
apply(drule deed_inv.cases; auto)
     apply(drule star_case; auto)
      apply(case_tac steps; auto)
      apply(split strict_if_split; auto)
      apply(split strict_if_split; auto)
```

Proof state Auto update Update Search:

```
proof (prove)
goal (7 subgoals):
 1. ⋀a initial_call aa bal origin ext block act st bala opt_v nat.
       account_code a = deed_program ⟹
       ucast (account_storage a 0) ≠ callenv_caller initial_call ⟹
       255 AND account_storage a 2 div 1461501637330902918203684832716283019655932542976 ≠ 0 ⟹
       account_balance a ≤ account_balance a + callenv_value initial_call ⟹
       ¬ account_killed a ⟹
       star (one_round deed_inv)
        (update_account_state a act st bala opt_v, ProgramToWorld (act, st, bala, opt_v))
        (aa, ProgramAnnotationFailure) ⟹
       bal (account_address a) = account_balance a ⟹
       (case strict_if (word_of_int (int (length (callenv_data initial_call))) ≠ 0)
             (blockedInstructionContinue
               (venv_advance_pc (cenv_program = deed_program, cenv_this = account_address a)
```

Output Query Sledgehammer Symbols

The proof finishes somehow

Isabelle2016 - Deed.thy

File Edit Search Markers Folding View Utilities Macros Plugins Help

Deed.thy (~/src/eth-isabelle/example/)

```
 apply(case_tac steps; auto)
apply(case_tac a; simp add: postcondition_pack_def add: deed_inv.simps)
done

text {* It takes 45 minutes to compile this proof on my machine.  Ten minutes
```

Proof state Auto update Update Search:

```
theorem
  deed_only_registrar_can_spend:
    pre_post_conditions deed_inv
     (λinit_state init_call.
         ucast (account_storage init_state 0) ≠ callenv_caller init_call ∧
         255 AND account_storage init_state 2 div 2 ^ 160 ≠ 0 ∧
         account_balance init_state ≤ account_balance init_state + callenv_value init_call ∧
         ¬ account_killed init_state)
     (λinit_state _ (post_state, uu_).
         account_balance init_state ≤ account_balance post_state ∧
         ¬ account_killed post_state ∧
         255 AND account_storage post_state 2 div 2 ^ 160 ≠ 0 ∧
         account_storage init_state 0 = account_storage post_state 0)
```

does this match the reality?

https://github.com/pirapira/eth-isabelle

Output Query Sledgeha

1174,1 (35415/35642)

If you can do better

1st Workshop on Trusted Smart Contracts

In Association with Financial Cryptography 17

April 07, 2017

The Palace Hotel & Spa

Malta