FORMAL LAND

COQ-OF-SOLIDITY

DECEMBER 2024

GENERAL PRESENTATION

FORMAL LAND

- **Formal verification company**
- **Dedicated to Web3**
- **Tezos, Aleph Zero, Sui, Ethereum**
- **Since 2021**

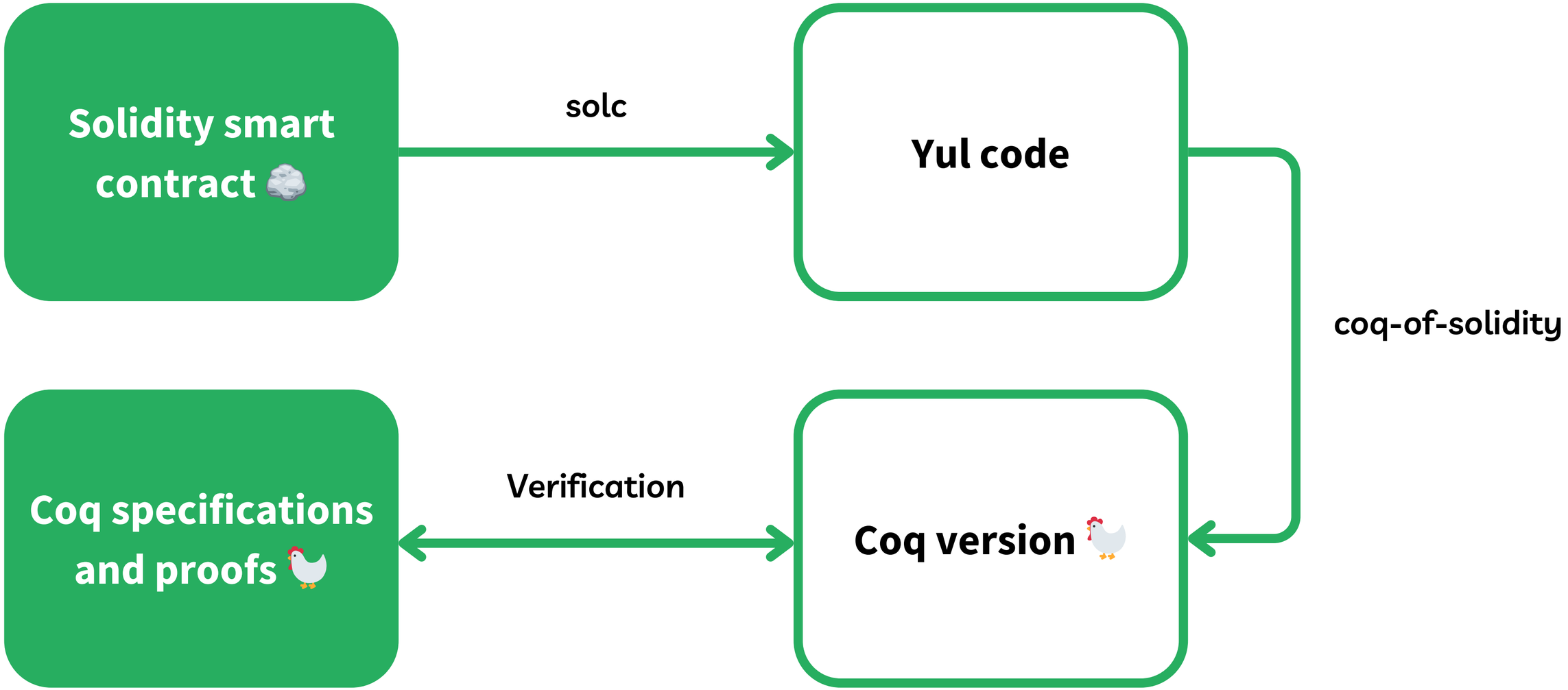
https://formal.land/

COQ-OF-SOLIDITY

A tool to verify smart contract on all possible inputs, and anticipate all vulnerabilities.

Open source:

https://github.com/formal-land/coq-of-solidity


Solidity smart contract
solc
Yul code
coq-of-solidity
Coq specifications and proofs
Verification
Coq version

SUPPORT

- **90%** of semantic tests of **solc**
- Missing:
  - Pre-compiles
  - Corner cases for contract calls

SEMANTICS

- **Memory** as an array of bytes
- No stack (Yul)
- **Blockchain state:** address → storage

SHALLOW EMBEDDING

- **Simpler** for proofs
- **Free monad** for primitives:
  - Storage
  - Unbounded loops
  - Contract calls
- **Error monad** for control flow (break, revert, ...)

MONADIC NOTATION

```
  |)
 ]] default~ (value0, value1) in
 let_state~ value0 :=
   let~ offset := [[ 0 ]] in
   let~ value0 := [[ abi_decode_t_uint256 ~(| add ~(| headStart, offs
   dataEnd |) ]] in
   M.pure (BlockUnit.Tt, value0)
 default~ (value0, value1) in
 let_state~ value1 :=
   let~ offset := [[ 32 ]] in
   let~ value1 := [[ abi_decode_t_uint256 ~(| add ~(| headStart, offs
   dataEnd |) ]] in
   M.pure (BlockUnit.Tt, value1)
 default~ (value0, value1) in
 M.pure (BlockUnit.Tt, (value0, value1))
in
M.pure (value0, value1).
```

TWO TRANSLATIONS

- **Raw translation** for compliance tests
- **Cleaned-up translation** for proofs

PROJECT

Generate a proof of equivalence between

these two translations

SEMANTIC RULES

- For the **free monad** primitives
- **Big-step**, by **continuation**
- To reason step by step on the code like in a **debugger**

SOURCE

```
//inlined ec_Add
T1:=shl(7, T1)//Commputed value address offset·····

let T4:=mload(add(Mem,T1))//X2
mstore(add(Mem, _zzz2), mload(add(Mem,add(96,T1))))//ZZZ2

if iszero(ZZ) {
  X := T4//X2
  Y := mload(add(Mem,add(32,T1)))//Y2
  ZZ := mload(add(Mem,add(64,T1)))//ZZ2
  ZZZ := mload(add(Mem,add(96,T1)))//ZZZ2
  continue
}
```

COQ TRANSLATION

```
let~ usrōT1 := [[ shl ~(| 7, usrōT1 |) ]] in
let~ usrōT4 := [[ mload ~(| add ~(| usrōMem, usrōT1 |) |) ]] in
do~ [[ mstore ~(| add ~(| usrōMem, 2144 |), mload ~(| add ~(| usrōMem, add ~(| 96, usrōT1 |) |) |) |) ]]
in
let_state~ '(var_X_60, var_Y_80, var_ZZZ_83, var_ZZ_86) := [[
  Shallow.if_ (|
    iszero ~(| var_ZZ_86 |),
    let~ var_X_60 := [[ usrōT4 ]] in
    let~ var_Y_80 := [[ mload ~(| add ~(| usrōMem, add ~(| 32, usrōT1 |) |) |) ]] in
    let~ var_ZZ_86 := [[ mload ~(| add ~(| usrōMem, add ~(| 64, usrōT1 |) |) |) ]] in
    let~ var_ZZZ_83 := [[ mload ~(| add ~(| usrōMem, add ~(| 96, usrōT1 |) |) |) ]] in
    M.pure (BlockUnit.Continue, (var_X_60, var_Y_80, var_ZZZ_83, var_ZZ_86)),
    (var_X_60, var_Y_80, var_ZZZ_83, var_ZZ_86)
  |)
]] default~ (var_X_60, var_Y_80, var_ZZZ_83, var_ZZ_86) in
```

COQ PROOF

```
(1/1)
{{?codes, environment,
Some
  (make_state environment state
     [mem0; mem1; 0; mem3; mem4; mem5; mem6; mem7; mem8; mem9; 480; mem11;
      mem12; mem13; mem14; Q.(PA.X); Q.(PA.Y); Q'.(PA.X);
      Q'.(PA.Y); p; a; G.(PA.X); G.(PA.Y); G'.(PA.X);
      G'.(PA.Y); 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0] [])
| let~ ' zero_t_uint256_1
  := M.call Contract_91.Contract_91_deployed.zero_value_for_split_t_uint256
  in let~ ' var_X_60 := pure zero_t_uint256_1
     in let~ ' expr_66 := pure 170141183460469231731687303715884105728
        in let~ ' var_mask_63
```

Command “l”

COQ PROOF

```
(1/1)
{{?codes, environment,
Some
  (make_state environment state
     [mem0; mem1; 0; mem3; mem4; mem5; mem6; mem7; mem8; mem9; 480; mem11;
      mem12; mem13; mem14; Q.(PA.X); Q.(PA.Y); Q'.(PA.X);
      Q'.(PA.Y); p; a; G.(PA.X); G.(PA.Y); G'.(PA.X);
      G'.(PA.Y); 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0] [])
| M.call Contract_91.Contract_91_deployed.zero_value_for_split_t_uint256
⇓ ?output_inter0 | ?state_inter0?}}
```

Command “c”

COQ PROOF

```
(1/1)
{{?codes, environment,
Some
  (make_state environment state
     [mem0; mem1; 0; mem3; mem4; mem5; mem6; mem7; mem8; mem9; 480; mem11;
      mem12; mem13; mem14; Q.(PA.X); Q.(PA.Y); Q'.(PA.X);
      Q'.(PA.Y); p; a; G.(PA.X); G.(PA.Y); G'.(PA.X);
      G'.(PA.Y); 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0] [])
| Contract_91.Contract_91_deployed.zero_value_for_split_t_uint256
⇓ ?output_inter1 | ?state_inter1?}}
```

Call to previous proof

DONE

- Functional specification of an ERC-20
- Verification of the beginning of an elliptic-curve multiplication library

GENERAL USE

1. Verify a **model** of the smart contract in Coq
2. Show it is equivalent to the **source** with coq-of-solidity

TESTING

We could also verify the model by testing

(integration with popular testing frameworks like

Foundry?)

BUSINESS

Proposing formal verification audits

FUTURE

- High-level primitives (identity, ownership, ...)
- Make a general statement to say that a smart contract is “safe” (no stealing/blocking)

THANKS

To get your smart contract formally verified:
contact@formal.land