FORMAL LAND

FORMAL VERIFICATION SOLIDITY IN COQ

OCTOBER 2024

ENCODE CLUB LONDON

FORMAL LAND

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

https://formal.land/

COQ-OF-SOLIDITY

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

Open source:

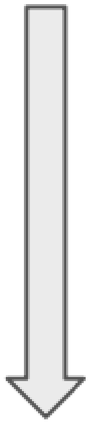
https://github.com/formal-land/coq-of-solidity

REASONS

1. **Billions** are stolen every year
2. **No police** on Web3
3. **Example:** Ethereum fork (DAO attack)

TECHNIQUES

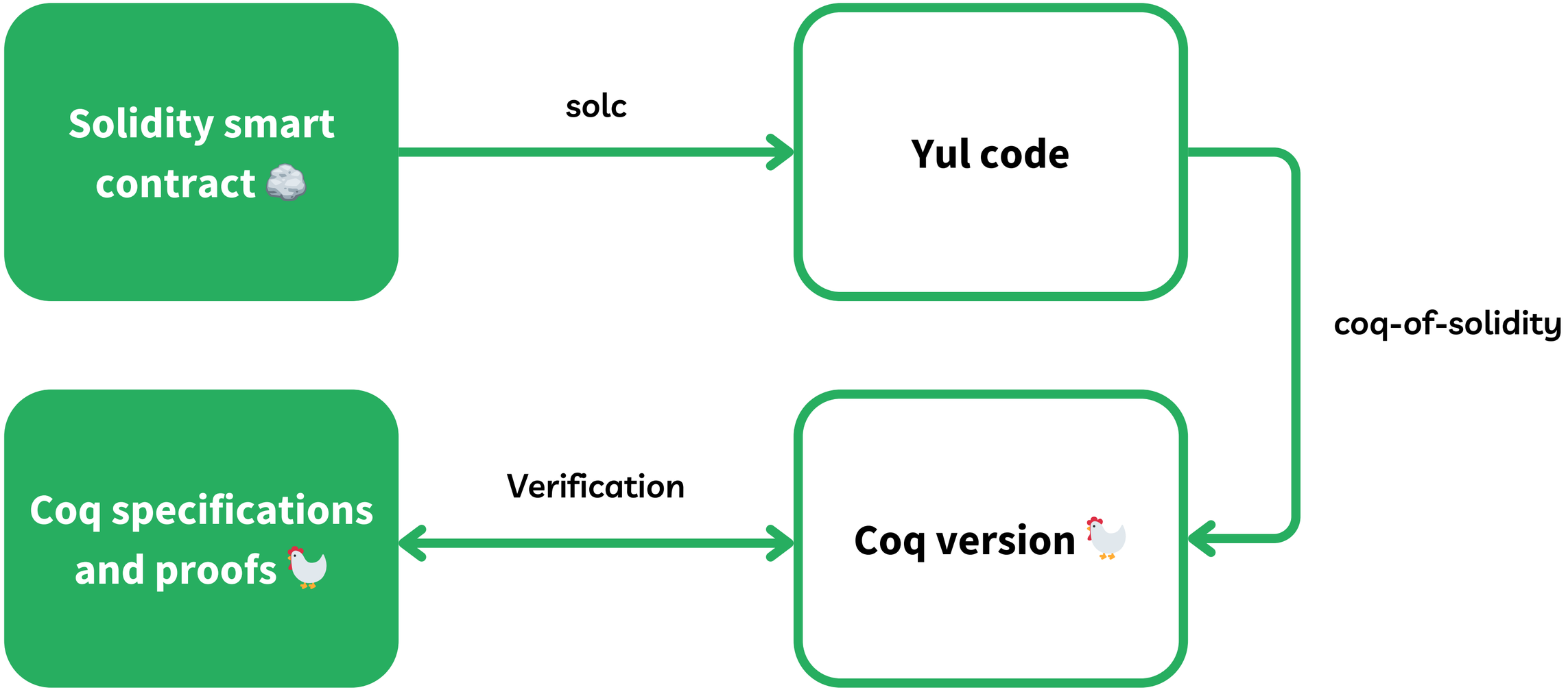
- Code review/Documentation
- Testing
- CI/CD
- Fuzzing/property based testing
- Modelling/Model Checking
- Symbolic Testing
- Program proof/Correct by Construction



Increasing effort,
Increasing confidence

Where we are

Diagram from a Galois' presentation


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

FORMAL VERIFICATION

- We can check **infinitely** many inputs
- In fact, we check all
- **How?**

HOW?

- We **do not execute** the code
- We reason on **symbols**

0 SPECIFICATIONS

- **What do we want to verify?**
- No blocked state
- No overflows
- Optimizations are correct

1 IF

if condition **then**

    action_then

else

    action_else

2 LOOP

for i = 0 to size-1 **do**

  action

done

3 FUNCTION

```
function foo() {
    x = bar()
    return x + 1
}
```

APPLICATION

- Verification of https://smoo.th/
- Optimized implementation of elliptic curves
- For authentication protocols
- Ongoing work

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~ usrõT1 := [[ shl ~(| 7, usrõT1 |) ]] in
let~ usrõT4 := [[ mload ~(| add ~(| usrõMem, usrõT1 |) |) ]] in
do~ [[ mstore ~(| add ~(| usrõMem, 2144 |), mload ~(| add ~(| usrõMem, add ~(| 96, usrõT1 |) |) |) |) ]]
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 := [[ usrõT4 ]] in
    let~ var_Y_80 := [[ mload ~(| add ~(| usrõMem, add ~(| 32, usrõT1 |) |) |) ]] in
    let~ var_ZZ_86 := [[ mload ~(| add ~(| usrõMem, add ~(| 64, usrõT1 |) |) |) ]] in
    let~ var_ZZZ_83 := [[ mload ~(| add ~(| usrõMem, add ~(| 96, usrõT1 |) |) |) ]] 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

WE GET

- A proof of equivalence with a functional version of the contract
- We proceed by refinements up to the original elliptic curve formula
- Still ongoing

THANKS

To get your smart contract verified:

contact@formal.land