Celestial: A Smart Contracts Verification Framework

FMCAD 2021

Samvid Dharanikota* · Suvam Mukherjee* · Chandrika Bhardwaj* · Aseem Rastogi · Akash Lal

Carnegie Mellon University · Microsoft Corporation · Goldman Sachs · Microsoft Research India

* work done at Microsoft Research India

Overview

Smart Contracts

- Programs executed on a blockchain
- Blockchain - distributed, immutable, append-only ledger
- Smart contracts cannot be patched/updated
- Diverse applications for smart contracts

Overview

Solidity & Ethereum Virtual Machine

- JavaScript-like OO language
- Compiles to EVM bytecode
- Each contract is a ‘class’
  - Contract state: Field variables
  - Contract logic: Functions
  - Data state part of contract
- Still under active development
- Every EVM instruction requires ‘gas’

Overview

The WrappedEther contract in Solidity:

```
contract WETH9 {
    event Deposit(address indexed dst, uint wad);

    mapping (address => uint) public balanceOf;

    function deposit() public payable {
        balanceOf[msg.sender] += msg.value;
        emit Deposit(msg.sender, msg.value);
    }
}
```


Overview

Bugs

- Solidity - obscure operational semantics
- Buggy smart contract logic => serious financial consequences
- Some common bugs:
  - SWC-101: Integer Overflow and Underflow
  - SWC-105: Unprotected Ether Withdrawal
  - SWC-107: Re-entrancy attacks
  - SWC-112: `delegatecall()` to Untrusted Callee
  - Denial of service in fallback

E.g.: The DAO Hack – led to loss of ~$70m.

Overview

Current Solutions

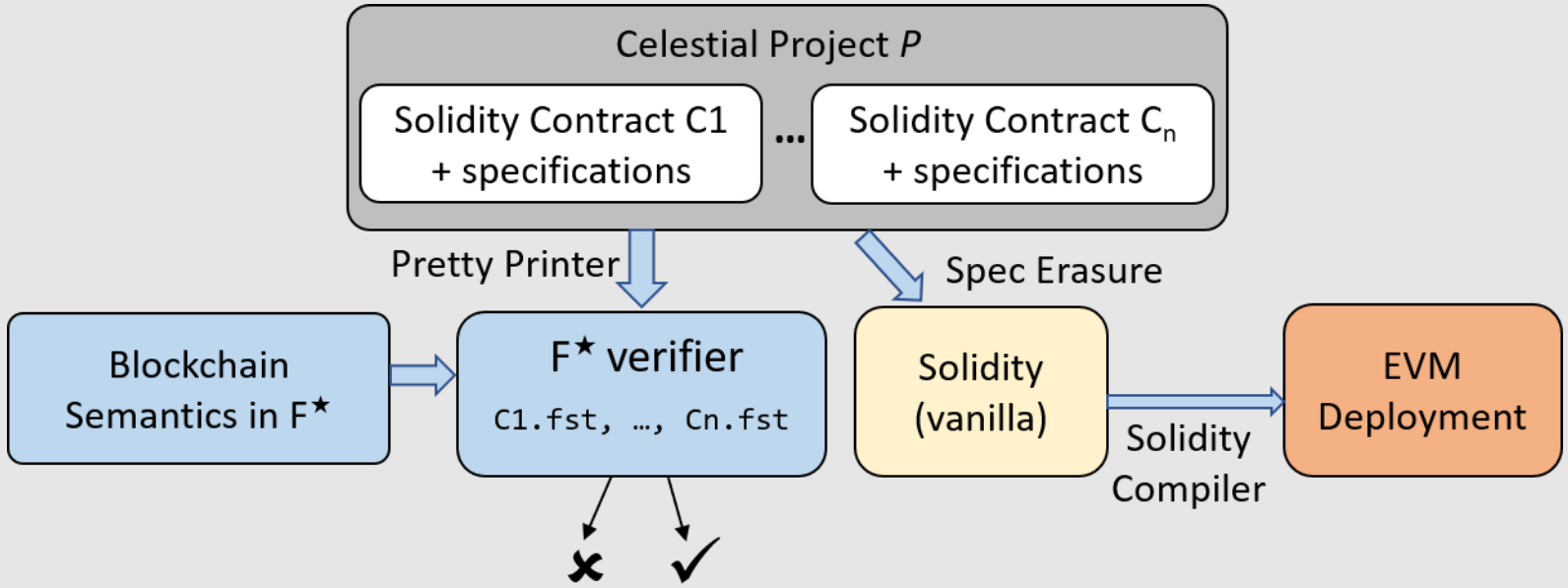
- Testing / Auditing / Static Analysis:
  - Oyente: https://oyente.melonport.com
  - Truffle tests: https://github.com/trufflesuite/truffle
  - https://contract-library.com/
  - MadMax: https://github.com/nevillegrech/MadMax
- Formal Verification:
  - Verisol: https://github.com/microsoft/verisol
  - K – Framework: https://github.com/kframework/evm-semantics
  - FSolidM / VeriSolid: https://github.com/anmavrid/smart-contracts

Drawbacks:

- Testing is inadequate
- Third-party auditing/analysis tools required

Our Solution: **Celestial**

- Framework for verification-driven smart contract development
  - Specification language to annotate Solidity
  - Verification backend (F*)
- F* backend verifies if contract conforms to its specification
- Spec erasure for execution on EVM



Celestial

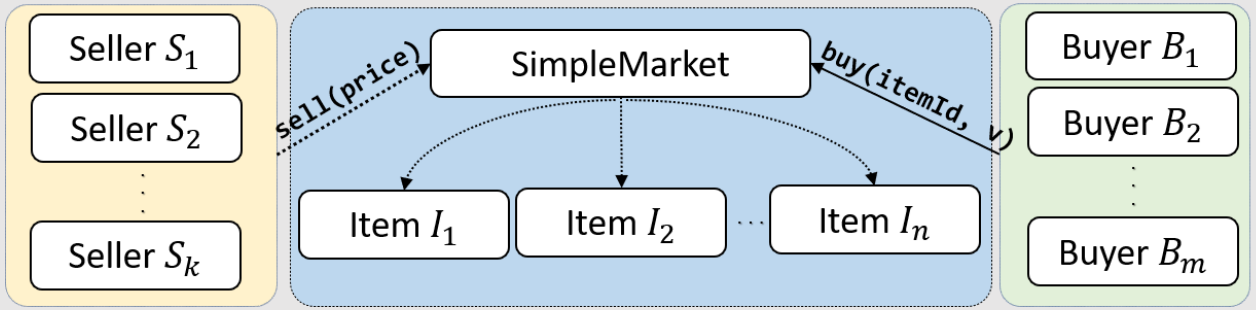
- Allows writing Hoare-style specifications:
  - pre
  - post
  - reverts
  - invariant
  - Other annotations (modifies, modifies_addresses, credit, debit)
- Fully automated verification
- Built-in runtime checks for safe arithmetic

```
Assume (pre /\ invariant)
Execute contract
Assert (!reverts ==> post /\ invariant) /\
       (reverts ==> exception)
```

```
contract A {
  uint x, y;
  invariant { Φ1 }
  function foo () public
    modifies [x]
    reverts Φ2
    pre Φ3
    post Φ4
  { s }
}
```


Celestial

Example contract




```
contract SimpleMarket {
  mapping(address => uint) sellerCredits;
  mapping(address => Item) itemsToSell;
  uint totalCredits ;
  event eNewItem (address, address);
  event eItemSold (address, address);
```

```
function buy (address itemId) public payable
returns (address seller) {
  Item item = itemsToSell[itemId];
  if (item == null)
    { revert("No such item "); }
  if (msg.value != item.getPrice())
    { revert(" Incorrect price "); }
  seller = item.getSeller();
  totalCredits = safe_add(totalCredits, msg.value);
  sellerCredits[seller] += msg.value ;
  delete(itemsToSell[itemId]);
  emit eItemSold (msg.sender, itemId);
}
```

Celestial

Contract Invariant and Specification for the `buy()` method

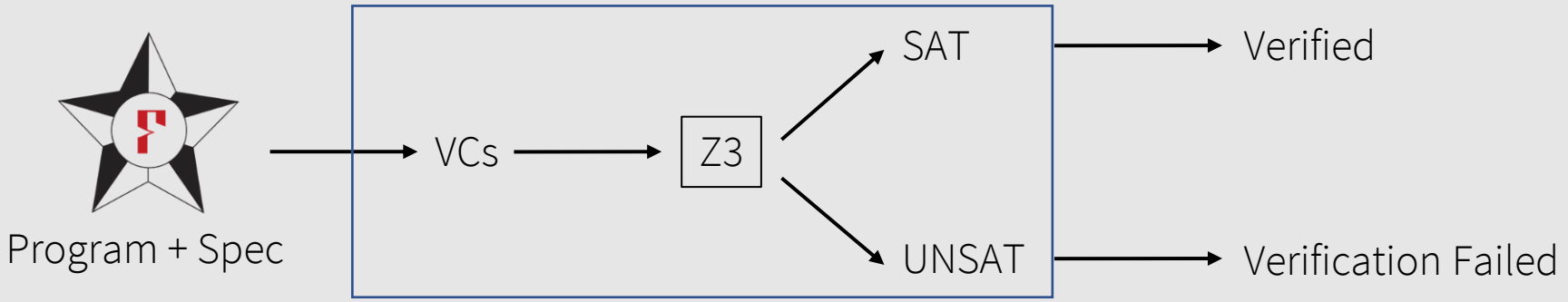
```
invariant balanceAndSellerCredits
{
  balance == totalCredits
  && totalCredits >= sum_mapping(sellerCredits)
}

function buy (address itemId) public returns (address seller)
modifies [sellerCredits, totalCredits, itemsToSell, log]
tx_reverts !(itemId in itemsToSell)
           || msg.value != itemsToSell[itemId].price
           || msg.value + totalCredits > uint_max
post (!(itemId in itemsToSell)
     && sellerCredits == old(sellerCredits)[seller => old(sellerCredits)[seller] + msg.value]
     && log == (eItemSold, msg.sender, itemId)::old(log))
{
  // implementation of the buy function
}
```

F*

- Functional programming language
  - Support for effects
  - Aimed at program verification

```
let divide (a:int) (b:int) : Pure int
  (requires b > 0)
  (ensures (fun r -> r == a / b))
= a / b
```



Solidity/EVM Model in F*

- Solidity/EVM semantics modelled in F*
- Celestial contracts verified against this model
- Fully-automated verification

Solidity/EVM Model in F*

Underlying types and functions

```
type address = uint                    (* 256 bit unsigned integers *)

val contract (a:Type) : Type           (* a is the record of contract fields *)

val cmap : Type                        (* a map from addresses to contracts *)

val live (#a:Type) (c:contract a) (m:cmap) : prop

val sel (#a:Type) (c:contract a) (m:cmap{live c m}) : a

val create (#a:Type) (m:cmap) (x:a) : contract a & cmap

val upd (#a:Type) (c:contract a) (m:cmap{live c m}) (x:a) : cmap

val addr_of (#a:Type) (c:contract a) : address
```


Solidity/EVM Model in F*

- Blockchain state (**bstate**) modelled as a tuple of:

```
type bstate {
  cmap    : Map.t address contract_t;
  balances: Map.t address uint;
  log     : list event
}
```

```
type state {
  tx_begin: bstate;
  current : bstate
}
```

- Effect (**Eth**):

```
effect ETH (a:Type) (pre:state -> prop) (post:state -> result a -> prop) = …

effect Eth (a:Type) (pre:bstate -> prop) (revert:bstate -> prop) (post:bstate -> a -> bstate -> prop)
= ETH a (λ s -> pre s.current)
        (λ s0 r s1 -> (revert s0.current ==> Error? x) /\
                      (Success? r ==> post s0.current (Success?.x r) s1.current))
```


Verification Scope and TCB

- Gas and EVM call-stack not modelled in F*
- No knowledge about actual state of contracts on Ethereum
- TCB:
  - Celestial compiler
  - F* blockchain model
  - F* toolchain
  - Solidity compiler

Experiments

- Verified several real world contracts from different domains (tokens, wallets, governance, auctions, etc)
  - Annotated these contracts with specs
  - Generated F* from Celestial
  - Verified generated F* for functional correctness

Experiments

- Case studies:
  - AssetTransfer: Azure Blockchain Workbench Contract
  - ConsenSys MultiSig: Multisignature wallet contract by ConsenSys ($19 million)
  - OpenZeppelin ERC20: Implementation of the ERC20 token standard by OpenZeppelin
  - BinanceCoin: An ERC20 token contract by Binance
  - WrappedEther: Another token contract ($16 billion)
  - Governance contract: Consortium management contract
  - SimpleAuction: Auction contract from Solidity docs
  - EtherDelta: Cross-token wallet ($45 million)

Experiments

- ERC20 Token:

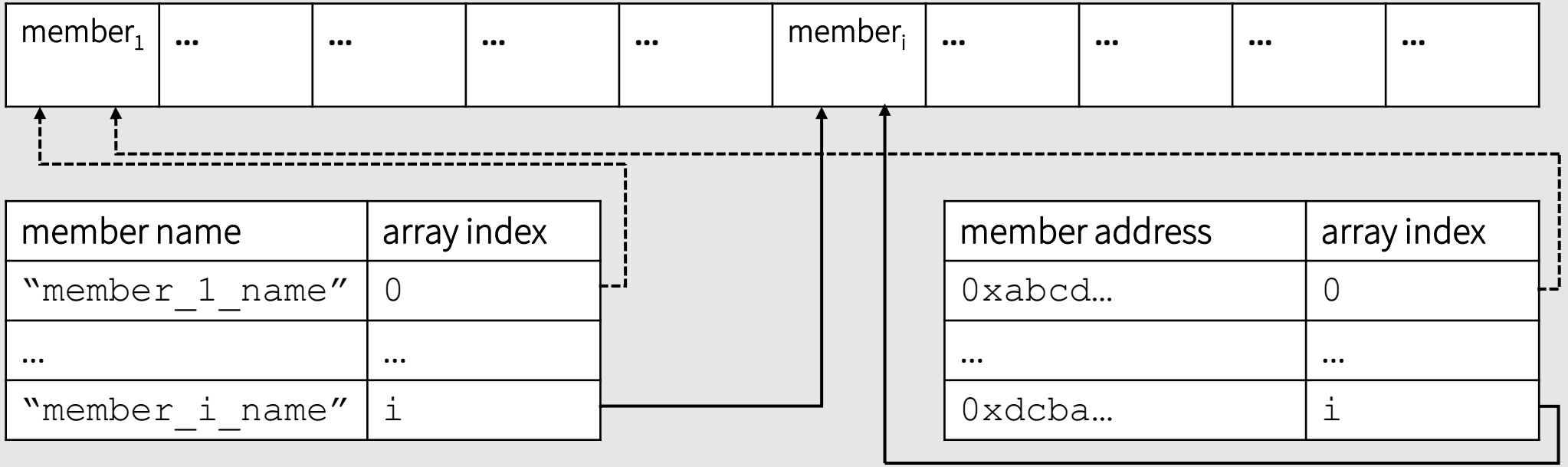
```
contract ERC20 {
 mapping(address => uint) _balances;
 uint _totalSupply;

 invariant _balanceAndSellerCredits {
   _totalSupply = sum_mapping(_balances)
 }

function _transfer (address from, address to, uint amt) private
 modifies [balances]
 pre _balances[from] >= amt
     && _balances[to] + amt <= uint_max
 post ite (
        from == to,
        _balances == old(_balances),
        _balances == old(_balances)[from => old(_balances)[from] - amt, to => old(_balances)[to] + amt]
      )
{ // implementation }
```


Experiments

- Governance Contract
  - Consortium Management – rules to add/remove members, etc.
  - Complex data structures:



Experiments

- Measured verification times:

Benchmark	Type	Spec LOC	Impl LOC	Verification Time (ms)
AssetTransfer	Marketplace	70	187	4.26
OpenZeppelin ERC20	Token	97	200	8.82
WrappedEther	Token	62	114	20.00
BinanceCoin	Token	25	136	29.98
EtherDelta	Wallet	57	351	63.97
ConsenSys MultiSig	MultiSig Wallet	163	289	77.80
SimpleAuction	Auction	61	101	22.45
GovernanceContract	Governance	121	149	86.86

Thank you

Celestial GitHub repository: https://github.com/microsoft/verisol/tree/celestial/Celestial

Contact: Samvid Dharanikota (samvid@cmu.edu)
Suvam Mukherjee (sumukherjee@microsoft.com)