Formal Verification of Smart Contracts

David J. Pearce

ConsenSys

@WhileyDave
whileydave.com

Overview

"Ethereum is a decentralized, open-source blockchain with **smart contract** functionality." –Wikipedia

"A **smart contract** is a computer program or a transaction protocol that is intended to automatically execute, control or document events and actions according to the terms of a contract or an agreement." –Wikipedia

Ethereum Virtual Machine (EVM)

Value	Mnemonic	δ	α	Description
0x00	STOP	0	0	Halts execution.
0x01	ADD	2	1	Addition operation. $\boldsymbol{\mu}'_{\mathbf{s}}[0] \equiv \boldsymbol{\mu}_{\mathbf{s}}[0] + \boldsymbol{\mu}_{\mathbf{s}}[1]$
0x02	MUL	2	1	Multiplication operation. $\boldsymbol{\mu}'_{\mathbf{s}}[0] \equiv \boldsymbol{\mu}_{\mathbf{s}}[0] \times \boldsymbol{\mu}_{\mathbf{s}}[1]$
0x03	SUB	2	1	Subtraction operation. $\boldsymbol{\mu}'_{\mathbf{s}}[0] \equiv \boldsymbol{\mu}_{\mathbf{s}}[0] - \boldsymbol{\mu}_{\mathbf{s}}[1]$
0x04	DIV	2	1	Integer division operation. $\boldsymbol{\mu}'_{\mathbf{s}}[0] \equiv \begin{cases} 0 & \text{if} \quad \boldsymbol{\mu}_{\mathbf{s}}[1] = 0 \\ \lfloor \boldsymbol{\mu}_{\mathbf{s}}[0] \div \boldsymbol{\mu}_{\mathbf{s}}[1] \rfloor & \text{otherwise} \end{cases}$

ETHEREUM: A SECURE DECENTRALISED GENERALISED TRA

BERLIN VERSION beacfbd – 2022-10-24

DR. GAVIN WOOD
FOUNDER, ETHEREUM & PARITY
GAVIN@PARITY.IO

Abstract. The blockchain paradigm when coupled with cryptographically-secured transac utility through a number of projects, with Bitcoin being one of the most notable ones. Each a simple application on a decentralised, but singleton, compute resource. We can call this paradigm a transactional singleton machine with shared-state.

Ethereum implements this paradigm in a generalised manner. Furthermore it provides a plurality of such resources, each with a distinct state and operating code but able to interact through a message-passing framework with others. We discuss its design, implementation issues, the opportunities it provides and the future hurdles we envisage.

1. Introduction

With ubiquitous internet connections in most places of the world, global information transmission has become incredibly cheap. Technology-rooted movements like Bitcoin have demonstrated through the power of the default, consensus mechanisms, and voluntary respect of the social contract, that it is possible to use the internet to make a decentralised value-transfer system that can be shared across the world and virtually free to use. This system can be said to be a very specialised version of a cryptographically secure, transaction-based state machine. Follow-up systems such as Namecoin adapted this original "currency application" of the technology into other applications, albeit rather simplistic ones.

Ethereum is a project which attempts to build the generalised technology; technology on which all transaction-based state machine concepts may be built. Moreover it aims to provide to the end-developer a tightly integrated end-to-end system for building software on a hitherto unexplored compute paradigm in the mainstream: a trustful object messaging compute framework.

1.1. **Driving Factors.** There are many goals of this project; one key goal is to facilitate transactions between consenting individuals who would otherwise have no means to trust one another. This may be due to geographical separation, interfacing difficulty, or perhaps the incompatibility, incompetence, unwillingness, expense, uncertainty, inconvenience, or corruption of existing legal systems. By specifying a state-change system through a rich and unambiguous language, and furthermore architecting a system such that we can reasonably expect that an agreement will be thus enforced autonomously, we can provide a means to this end.

Dealings in this proposed system would have several

is often lacking, and plain old prejudices are difficult to shake.

Overall, we wish to provide a system such that users can be guaranteed that no matter with which other individuals, systems or organisations they interact, they can do so with absolute confidence in the possible outcomes and how those outcomes might come about.

1.2. **Previous Work.** Buterin [2013a] first proposed the kernel of this work in late November, 2013. Though now evolved in many ways, the key functionality of a blockchain with a Turing-complete language and an effectively unlimited inter-transaction storage capability remains unchanged.

Dwork and Naor [1992] provided the first work into the usage of a cryptographic proof of computational expenditure ("proof-of-work") as a means of transmitting a value signal over the Internet. The value-signal was utilised here as a spam deterrence mechanism rather than any kind of currency, but critically demonstrated the potential for a basic data channel to carry a *strong economic signal*, allowing a receiver to make a physical assertion without having to rely upon *trust*. Back [2002] later produced a system in a similar vein.

The first example of utilising the proof-of-work as a strong economic signal to secure a currency was by Vishnumurthy et al. [2003]. In this instance, the token was used to keep peer-to-peer file trading in check, providing "consumers" with the ability to make micro-payments to "suppliers" for their services. The security model afforded by the proof-of-work was augmented with digital signatures and a ledger in order to ensure that the historical record couldn't be corrupted and that malicious actors could not spoof payment or unjustly complain about service delivery. Five years later, Nakamoto [2008] introduced another such proof-of-work-secured value token, somewhat wider in scope. The fruits of this project, Bitcoin, became the first

Ethereum Virtual Machine (EVM)

```
PUSH1 0x10
MLOAD
PUSH1 0x20
SLOAD
ADD
PUSH1 0x20
SSTORE
```

- **Stack**. For instruction operands.
- **Memory**. Temporary for contract call.
- **Storage**. Persistent across contract calls.

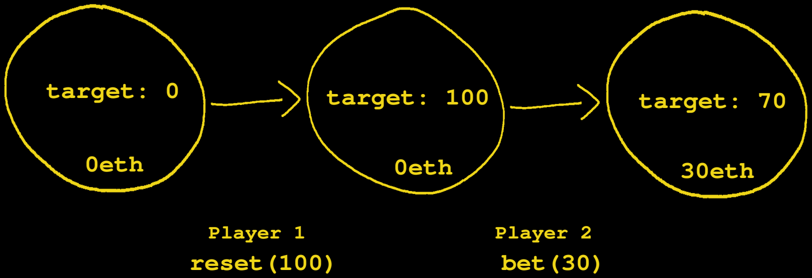
Solidity

```
contract Betting {
  uint public target = 0;

  function bet() public payable {
    require(msg.value <= target);
    unchecked { target = target - msg.value; }
    if(target == 0) {
      payable(msg.sender).transfer(address(this).balance);
    }
    assert target == 0;
  }

  function reset(uint newTarget) public {
    require(newTarget <= 1 ether);
    require(target == 0);
    target = newTarget;
} }
```

Betting Contract: State Transition Diagram



Solidity: Modifiers

```
modifier onlyOwner {
  require(msg.sender == owner);
  _;
}

function f() onlyOwner {
  ...
}
```

- Can be used to enforce global **correctness properties**
- Sadly, can do other things (e.g. **having effects**).

Solidity: Deposit Contract

```
deposit(...)
{
  while C1 do
  if C2 return;
     ...
  od
  // As the loop should always end prematurely with the ‘return‘
  // statement, this code should be unreachable. We assert ‘false‘
  // just to be safe.
  assert (false);
}
```

–Cassez, *et al.*, FM'21

(contract currently holds around 9million ETH)

Token Contract

Token Contract: Solidity

```
contract Token {
  address owner;
  mapping(address=>uint) tokens;
  uint total;

  constructor() { owner = msg.sender; }

  function mint(address acct, uint amount) public onlyOwner {
    tokens[acct] = amount;
    total = total + amount;
  }

  function transfer(address to, uint amount) public {
    tokens[to] += amount;
    tokens[msg.sender] -= amount;
} }
```

Token Contract: Dafny

```
class Token {
  var owner: address;
  var tokens: map<address,uint>;
  var total: uint;

  constructor() { ... }

  method mint(account: address, amount: uint)
  requires msg_sender == owner {
    ...
  }
  method transfer(to: address, amount: uint)
  returns (ok:bool) {
    ...
} }
```

Token Contract: Sum of Balances

A key property of the token contract is that `total` *equals the sum over all balances.*

Reentrancy

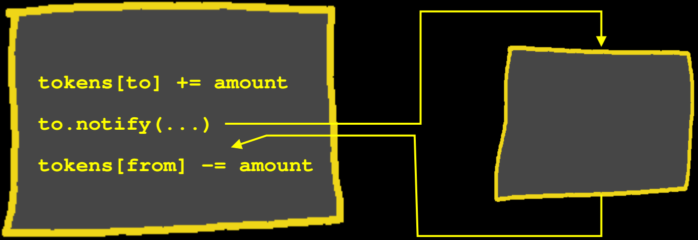
"... **other contracts** *are typically developed by unknown parties and cannot be assumed to be verified; they might even exhibit adversarial behaviour to gain a financial advantage. As a result, standard modular reasoning techniques such as separation logic, which reason about calls under the assumption that all code is verified,* **do not apply in this setting***."*

– Bräm, et al., OOPSLA'21

Token Contract: Reentrancy

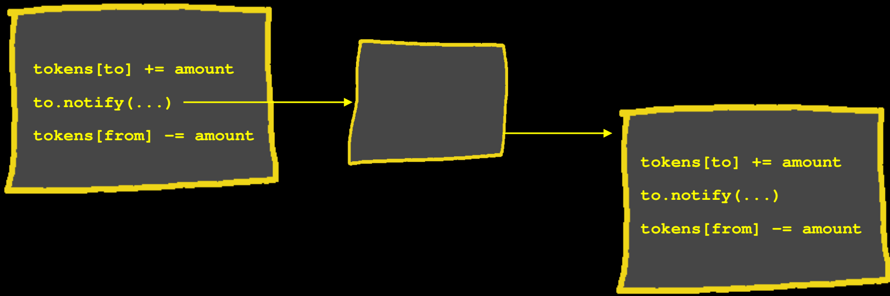
```
function transfer(address to, uint amount) public {
  tokens[to] += amount;
  to.call{gas: 5000}(abi.encodeWithSignature("notify()"));
  tokens[msg.sender] -= amount;
}
```

Token Contract: The Happy Path



`{ sum(tokens) == old(sum(tokens)) }`   `{ tokens == old(tokens) }`

Token Contract: The Not So Happy Path



Token Contract: Fixed!

```
function transfer(address to, uint amount) public {
  tokens[to] += amount;
  tokens[msg.sender] -= amount;
  to.call{gas: 5000}(abi.encodeWithSignature("notify()"));
}
```

Token Contract: Another Solution

```
function transfer(address to, uint amount) public {
  if(!locked) {
    tokens[to] += amount;
    locked = true;
    to.call{gas: 5000}(...);
    locked = false;
    tokens[msg.sender] -= amount;
  }
}
```

Bytecode Verification

Bytecode Verification: Example

```
const BYTECODE := [
    PUSH1,x,
    PUSH1,y,
    ADD
];

method add_bytes(x: u8, y: u8) {
  var st := InitEmpty(gas:=1000, code:=BYTECODE);
  st := Execute(st); // PUSH1
  st := Execute(st); // PUSH1
  st := Execute(st); // ADD
  assert st.Peek(0) == (x as u256) + (y as u256);
}
```

http://whiley.org

@WhileyDave
http://github.com/Whiley