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)

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

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

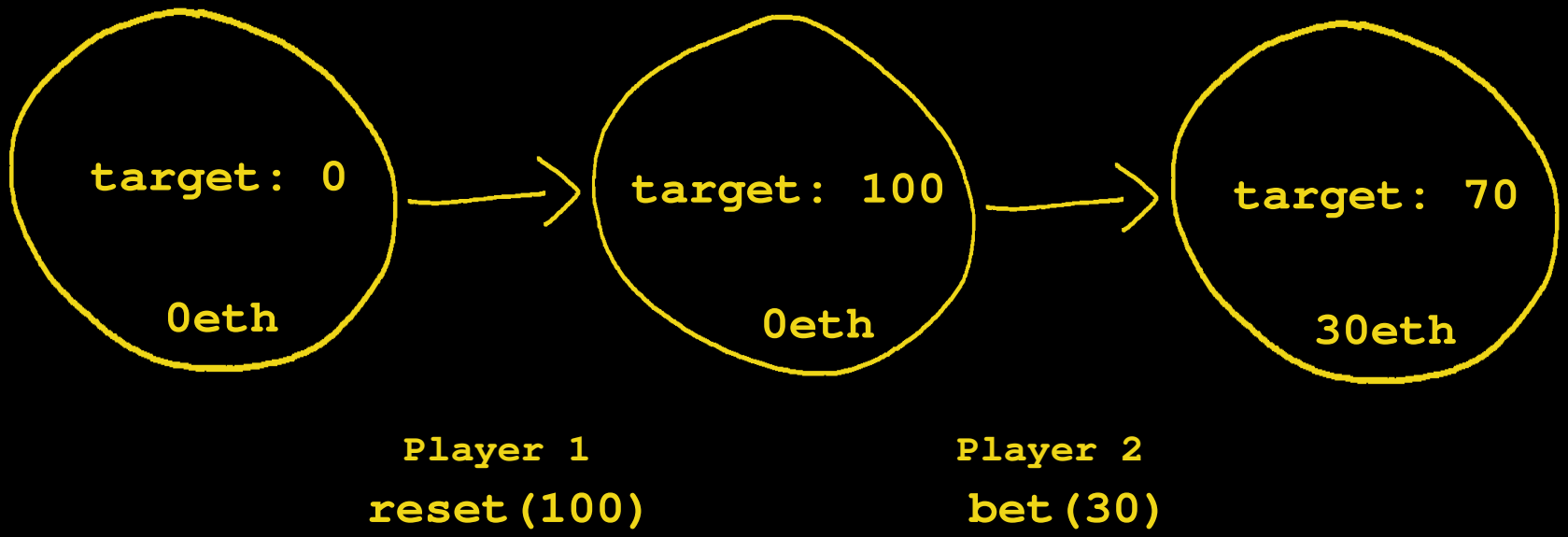
Solidity

```
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**).

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

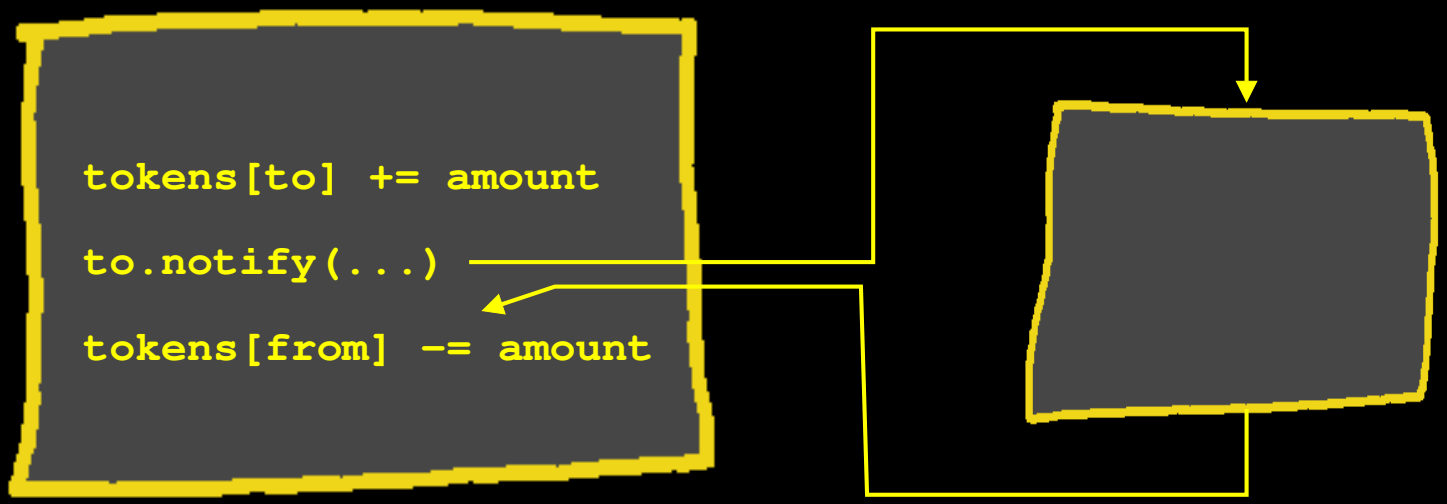
"... **other contracts** *are typically developed by unknown parties and cannot be assumed to be verified; they might even exhibit adversarial behaviour to gain a financial advantage. As a result, standard modular reasoning techniques such as separation logic, which reason about calls under the assumption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

– Bräm, et al., OOPSLA'21

Token Contract: Reentrancy

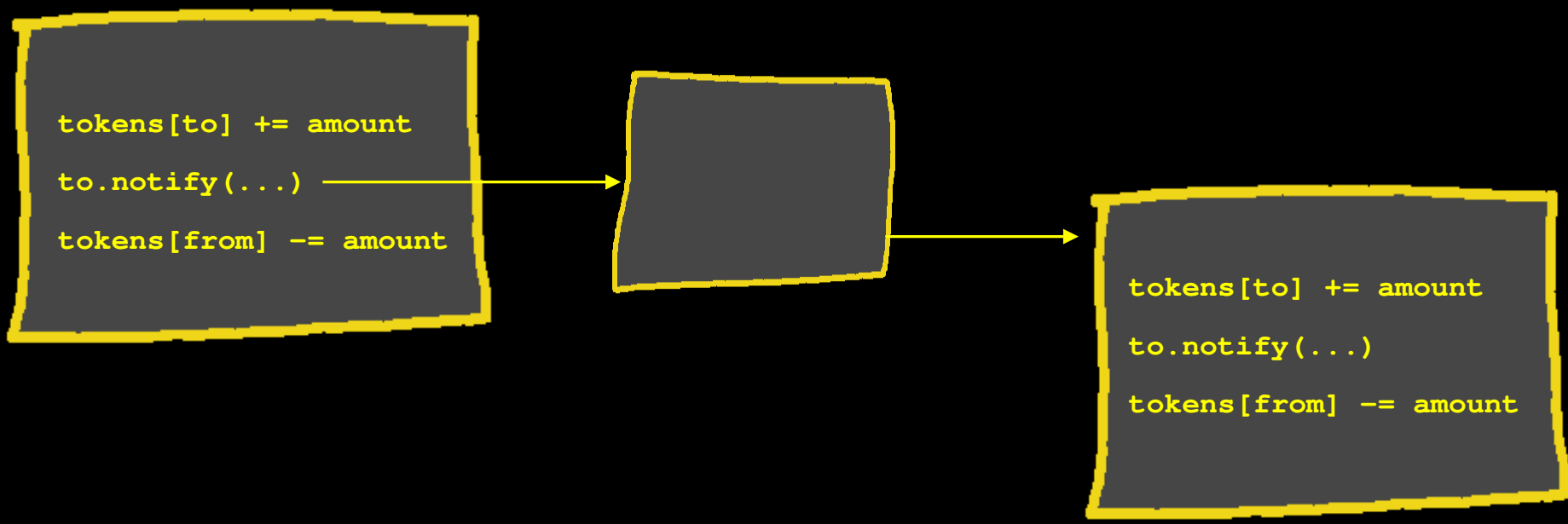
```
function transfer(address to, uint amount) public {
  tokens[to] += amount;
  to.call{gas: 5000}(abi.encodeWithSignature("notify()"));
  tokens[msg.sender] -= amount;
}
```

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