Updates from the Solidity team

Erik Kundt

Ethereum Foundation

erak

erik@ethereum.org

Agenda

1. Team
2. Audits
3. Solidity 0.5.0
4. Upcoming
5. How to contribute
6. Q & A

Agenda

1. Team
2. Audits
3. Solidity 0.5.0
4. Upcoming
5. How to contribute
6. Q & A

Team

- Alex Beregszaszi (@axic)
- Christian Parpart (@christianparpart)
- Christian Reitwiessner (@chriseth)
- Chris Ward (@chrischinchilla)
- Daniel Kirchner (@ekpyron)
- Erik Kundt (@erak)
- Leonardo Alt (@leonardoalt)

Agenda

1. Team
2. Audits
3. Solidity 0.5.0
4. Upcoming
5. How to contribute
6. Q & A

Audits

1st Audit:

- End of 2017 by Coinspect for Augur
- Discovered 10 issues, fixed 9 and the last issue is part of the inheritance changes
- Report: https://medium.com/@AugurProject/solidity-compiler-audit-report-1832cedb50a8

Audits

2nd Audit:

- Started in June by Zeppelin
- Sponsored by EF / Augur
- Working closely with the Solidity team
- Auditing a specific compiler version, but issues are being fixed in develop already (and most of them will be part of 0.5.0)

Audits

Talk about Solidity audit:

- **Solidity Compiler Audit Post-mortem**
- When: Day 2, 1:40PM
- Where: Ultra Violet (Breakout Room)
- Who: Manuel Araoz (OpenZeppelin)

Agenda

1. Team
2. Audits
3. Solidity 0.5.0
4. Upcoming
5. How to contribute
6. Q & A

Design goals

Safety through:

- Requiring users to be more explicit
- Removing disambiguities or weird behavior
- Adding run-time checks

Language features

- Explicit types
- Explicit visibility
- Explicit data locations
- Scoping rules for function local variables
- New constructor syntax
- emit for events
- address payable
- Others

Explicit types

Explicit types

```
contract Old {
  function f() public {
    for (var i = 0; i < 256; i++) {
      // Will this ever finish?
    }
  }
}
```

Explicit types

var is disallowed:

```
contract New {
  function f() public {
    for (uint i = 0; i < 256; i++) {
      // Yes, it will.
    }
  }
}
```

Explicit visibility

Explicit visibility

```
contract Old {
  address owner;
  function Old() {
    initialize();
  }
  function initialize() {
    owner = msg.sender;
  }
  function withdraw() {
    require(msg.sender == owner);
    msg.sender.transfer(address(this).balance);
  }
}
```

Explicit visibility

Visibility specifier is mandatory:

```
contract New {
  address owner;
  function New() public {
    initialize();
  }
  function initialize() internal {
    owner = msg.sender;
  }
  function withdraw() public {
    require(msg.sender == owner);
    msg.sender.transfer(address(this).balance);
  }
}
```

Explicit data locations

Explicit data locations

```
contract Old {
    struct Data { string name; }
    Data[] members;

    function f(uint index) public {
        Data member = members[index];
    }
}
```

Explicit data locations

Location specifier is mandatory:

```
contract New {
    struct Data { string name; }
    Data[] members;

    function f(uint index) public {
        Data storage member = members[index];
    }
}
```

Storage references

```
contract Old {
    struct Data { string name; }
    Data[] members;

    function f(string name) public {
        Data member;
        member.name = name;
        members.push(member);
    }
}
```

Storage references

Storage references have to be initialized:

```
contract New {
    struct Data { string name; }
    Data[] members;

    function f(string memory name) public {
        uint index = members.length;
        members.length++;
        Data storage member = members[index];
        member.name = name;
    }
}
```

Scoping rules

Scoping rules

Function-scoped variables (JavaScript):

```
contract Old {
  function f() pure public returns(uint) {
    i = 3;
    if (false)
      uint i;
    return i;
  }
}
```

Scoping rules

Block-scoped variables (C99-style):

```
contract New {
  function f(uint[] memory a, uint[] memory b) pure public returns(uint sum) {
    for (uint i; i < a.length; i++) {
      sum += a[i];
    }
    for (uint i; i < b.length; i++) {
      sum += b[i];
    }
  }
}
```

New `constructor` syntax

New constructor syntax

```
contract Old {
  address owner;
  function New() public {
    initialize();
  }
  function initialize() internal {
    owner = msg.sender;
  }
}
```

New constructor syntax

```
contract New {
  address owner;
  constructor() public {
    initialize();
  }
  function initialize() internal {
    owner = msg.sender;
  }
}
```

`emit` for events

emit for events

```
contract Old {
  address owner;
  event Withdrawn();
  function withdraw() public {
    require(msg.sender == owner);
    msg.sender.transfer(address(this).balance);
    Withdrawn();
  }
}
```

emit for events

Event invocations must be prefixed:

```
contract New {
  address owner;
  event Withdrawn();
  function withdraw() public {
    require(msg.sender == owner);
    msg.sender.transfer(address(this).balance);
    emit Withdrawn();
  }
}
```

address payable

address payable

```
contract Old {
  function f() public {
    address target = 0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c;
    target.transfer(1 ether);
  }
}
```

address payable

Address payable is required:

```
contract New {
  function f() public {
    address payable target = 0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c;
    target.transfer(1 ether);
  }
}
```

address payable

```
contract Old {
  function f() public {
    address payable target = address(this);
    target.transfer(1 ether);
  }
}
```

address payable

this needs explicit conversion:

```
contract C {
  function f() public {
    address payable target = address(this);
    target.transfer(1 ether);
  }
  function() external payable { }
}
```

Others

- abi.encode() and abi.decode()
- call, delegatecall, keccak256, etc. take a single parameter
- view / pure functions use staticcall
    - library view functions use delegatecall

Agenda

1. Team
2. Audits
3. Solidity 0.5.0
4. Upcoming
5. How to contribute
6. Q & A

Upcoming

- Yul
- Formal verification: SMTChecker
- Inheritence rule changes
- Contract metadata
- ABI encoder V2

Yul

Yul

- Assembly language (IR)
- Aids auditing of the codebase and generated code
- Allows optimization
- Allows multiple backends:
    - Solidity -> Yul -> EVM
    - Solidity -> Yul -> ewasm
    - Vyper -> Yul -> EVM or ewasm
- Read more:

  https://solidity.readthedocs.io/en/develop/yul.html

Yul

```
function power(base:u256, exponent:u256) -> result:u256
{
  switch exponent
  case 0:u256 { result := 1:u256 }
  case 1:u256 { result := base }
  default
  {
    result := power(mul(base, base), div(exponent, 2:u256))
    switch mod(exponent, 2:u256)
      case 1:u256 { result := mul(base, result) }
    }
  }
}
```

Yul

Talk about Yul and it's optimizer:

- **Less Gas, More Fun: Optimising Smart Contracts through Yul**
- When: Day 1, 4:30PM
- Where: Prism ("Side Stage")
- Who: Christian Reitwiessner (Solidity)

Formal verification

Formal verification

- SMT (satisfiability modulo theories)
- Seamless verification of safety properties:
  - Overflow
  - Underflow
  - Division by zero
  - Trivial conditions & unreachable code
  - Assertions (verifying runtime checks at compile time)
- Component in the compiler: SMTChecker

Formal verification

```
pragma experimental SMTChecker;
contract C {
  function f(uint a) returns (uint) {
    for (uint i = 200; i >= 0; i--) {
    }
  }
}
```

Formal verification

```
pragma experimental SMTChecker;
contract C {
  function f(uint a) returns (uint) {
    for (uint i = 200; i >= 0; i--) {
    }
  }
}
```

```
underflow.sol:4:28: Warning: For loop condition is always true.
  for (uint i = 200; i >= 0; i--) {
                     ^____^
underlow.sol:4:36: Warning: Underflow (resulting value less than 0) happens here
  for (uint i = 200; i >= 0; i--) {
                             ^_^
```

Formal verification

```
pragma experimental SMTChecker;
library SafeMath {
  function add(uint a, uint b) public returns (uint c) {
    c = a + b;
    assert(c >= a);
    return a;
  }
}
```

Formal verification

```
pragma experimental SMTChecker;
library SafeMath {
  function add(uint a, uint b) public returns (uint c) {
    c = a + b;
    assert(c >= a);
    return a;
  }
}
```

```
safemath.sol:4:9: Warning: Overflow (resulting value larger than 0xffffffffffffffffffffffffffffff...) happens here
  c = a + b;
  ^-------^
```

Formal verification

Talk about the SMTChecker:

- **Using Solidity's SMTChecker**
- When: Day 2, 4:00PM
- Where: Ultra Violet (Breakout Room)
- Who: Leonardo Alt (Solidity)

Inheritence rule changes

Inheritence rule changes

- Crucial part of Solidity contracts
- What about explicit shadowing?
- Can visibility / state mutability levels change?
- Lets design a more cohesive set of rules: https://github.com/ethereum/solidity/pull/3729

Contract metadata

Contract metadata

- Generated for each contract (as a JSON object)
- All details needed to reproduce compilation are included
- Hash of this metadata is appended to the bytecode
- Not used by verification tools yet
- See: https://solidity.readthedocs.io/en/v0.4.24/metadata.html

ABI encoder V2

ABI encoder V2

V1:

- "Contract ABI" is the specification how to exchange data with a contract
- For each public function a decoder/encoder is generated
- Handwritten generator of EVM bytecode in C++

ABI encoder V2

V2:

- Written in Yul (EVM assembly language)
- Ensures safety properties (short input, invalid values)
- Complex data types: structs, multi-dimensional arrays
- Try it out:

```
pragma experimental ABIEncoderV2;
```

Agenda

1. Team
2. Complex data types: structs, multi-dimensional arrays
3. Audits
4. Solidity 0.5.0
5. Upcoming
6. How to contribute
7. Q & A

How to contribute

- Many ways to contribute:
  - Feature requests
  - Discussions on existing design issues
  - Documentation improvements (including examples)
- Start with issues labeled "help wanted" or "good first issue" on Github
- Watch out for Gitcoin bounties!
- Reach out on https://gitter.im/ethereum/solidity-dev

Agenda

1. Team
2. Audits
3. Solidity 0.5.0
4. Upcoming
5. How to contribute
6. Q & A

Questions?

Thank you!

Formal verification: Example

De Morgan's law:

```
!(x && y) == (!x || !y)
```

Prove that

```
not(x and y) <-> (not x) or (not y)
```

holds.

Contract metadata

```
{
  "version": 1,
  "language": "Solidity",
  "compiler": {
      "version": "0.4.25+commit.59dbf8f1"
  },
  "settings": {
      "compilationTarget": {
          "test.sol": "Test"
      },
      "evmVersion": "byzantium",
      "optimizer": {
          "enabled": "false"
      }
  },
  "sources": {
      "test.sol": {
          "keccak256": "0x433008f6c5fdb9e9becb3999b296f2fe5f7836c88727e4eac09bb7d8e909d05",
          "urls": [
              "bzzr://c5627765490a3fb8de578bc27714fb87fb8b8d16b88c8dd06ac66a113328fd"
          ]
      }
  }
}
```