



Fully Automated Formal Verification: How far can we go?

Leo Alt & Martin Lundfall

Ethereum Foundation

🔗 leonardoalt, MrChico

✉ {leo, martin.lundfall}@ethereum.org

🐦 leonardoalt, MartinLundfall

```

contract Token {
    mapping (address => uint) balance;
    address immutable owner;

    constructor(uint _amt) {
        owner = msg.sender;
        balance[msg.sender] = _amt;
    }

    function balanceOf(address _user) public view returns (uint) {
        return balance[_user];
    }

    function transfer(address _to, uint _amt) public {
        require(msg.sender != _to);
        require(balance[msg.sender] >= _amt);

        uint prevBal = balance[msg.sender] + balance[_to];

        balance[msg.sender] -= _amt;
        balance[_to] += _amt;

        uint postBal = balance[msg.sender] + balance[_to];

        assert(prevBal == postBal);
    }
}

```

```

contract AMM is ERC20 {
    ERC20 token0;
    ERC20 token1;

    constructor(address _token0, address _token1) {
        token0 = ERC20(_token0);
        token1 = ERC20(_token1);
    }

    // swap allows the caller to exchange amt of src for dst at a price given
    // by the constant product formula: x * y == k.
    function swap(address src, address dst, uint amt) external {
        require(src != dst, "no self swap");
        require(src == address(token0) || src == address(token1), "src not in pair");
        require(dst == address(token0) || dst == address(token1), "dst not in pair");

        uint k = token0.balanceOf(address(this)) * token1.balanceOf(address(this));

        ERC20(src).transferFrom(msg.sender, address(this), amt);

        uint out =
            ERC20(dst).balanceOf(address(this)) -
            (k / ERC20(src).balanceOf(address(this)));

        ERC20(dst).transfer(msg.sender, out);

        uint kpost = token0.balanceOf(address(this)) * token1.balanceOf(address(this));

        assert(kpost >= k);
    }
}

```



```

function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
    require(recipient != address(0), "ERC777: transfer to the zero address");
    address from = _msgSender();
    _callTokensToSend(from, from, recipient, amount);
    _move(from, from, recipient, amount);
    _callTokensReceived(from, from, recipient, amount);
    return true;
}

/**
 * @dev Call from.tokensToSend() if the interface is registered
 * @param operator address operator requesting the transfer
 * @param from address token holder address
 * @param to address recipient address
 * @param amount uint256 amount of tokens to transfer
 */
function _callTokensToSend(address operator, address from, address to, uint256 amount) private {
    uint prevTotal = _totalSupply;
    IERC777Sender(from).tokensToSend(operator, from, to, amount);
    assert(prevTotal == _totalSupply);
}

/**
 * @dev Call to.tokensReceived() if the interface is registered. Reverts if the recipient is a contract but
 * tokensReceived() was not registered for the recipient
 * @param operator address operator requesting the transfer
 * @param from address token holder address
 * @param to address recipient address
 * @param amount uint256 amount of tokens to transfer
 */
function _callTokensReceived(address operator, address from, address to, uint256 amount) private {
    uint prevTotal = _totalSupply;
    IERC777Recipient(to).tokensReceived(operator, from, to, amount);
    assert(prevTotal == _totalSupply);
}

```

- 🔹 Mythril – symbolic execution
- 🔹 hevm – symbolic execution, invariant fuzzing
- 🔹 Echidna – invariant fuzzing
- 🔹 SMTChecker – model checking
- 🔹 solc-verify – model checking
- 🔹 VeriSmart – model checking

Tools that claim to try to prove/break properties automatically
and are publicly available.

Which tool can either **prove correctness** or **find bugs** in all the examples?

Automated formal verification is undecidable

Target Solidity

Cons

- ❖ A lot to encode: high level features, various data types, pointers, inheritance.
- ❖ Results rely on compiler correctness.

Pros

- ❖ Gives more structure information, for example, loops, external calls, functions.
- ❖ Can try harder problems, involving loop/contract invariants.

Model checking: SMTChecker, solc-verify, VeriSmart

Target EVM bytecode

Cons

- ⚠ Not a lot of structure.
- ⚠ Hard to track storage, external calls, functions.
- ⚠ Needs to verify ABI encoding/decoding.

Pros

- ⚠ Easier to encode.
- ⚠ Results are closer to the deployed object.

Symbolic execution and fuzzing: Mythril, hevm, Echidna

Experiment:

Use each tool on each example, first automatically then tweaking parameters and writing specs taylored to the tool.

Prove functional correctness of transfer function

🔹 Mythril - OK

🔹 hevm - OK

🔹 Echidna - OK (no bugs found)

🔹 SMTChecker - OK

🔹 solc-verify - OK

🔹 VeriSmart - OK

Find bug in buggy transfer function

```
contract Token {
    mapping (address => uint) balance;
    address immutable owner;

    constructor(uint _amt) {
        owner = msg.sender;
        balance[msg.sender] = _amt;
    }

    function balanceOf(address _user) public view returns (uint) {
        return balance[_user];
    }

    function transfer(address _to, uint _amt) public {
        require(msg.sender != _to);
        require(balance[msg.sender] >= _amt);

        uint prevBal = balance[msg.sender] + balance[_to];

        balance[msg.sender] += _amt;
        balance[_to] += _amt;

        uint postBal = balance[msg.sender] + balance[_to];

        assert(prevBal == postBal);
    }
}
```

Find bug in buggy transfer function

- Mythril – OK, with counterexample
- hevm – OK, with counterexample
- Echidna – OK, with counterexample
- SMTChecker – OK, with counterexample
- solc-verify – OK, no counterexample
- VeriSmart – No

AMM swap functional correctness (?)

```
contract AMM is ERC20 {
    ERC20 token0;
    ERC20 token1;

    constructor(address _token0, address _token1) {
        token0 = ERC20(_token0);
        token1 = ERC20(_token1);
    }

    // swap allows the caller to exchange amt of src for dst at a price given
    // by the constant product formula: x * y == k.
    function swap(address src, address dst, uint amt) external {
        require(src != dst, "no self swap");
        require(src == address(token0) || src == address(token1), "src not in pair");
        require(dst == address(token0) || dst == address(token1), "dst not in pair");

        uint k = token0.balanceOf(address(this)) * token1.balanceOf(address(this));

        ERC20(src).transferFrom(msg.sender, address(this), amt);

        uint out =
            ERC20(dst).balanceOf(address(this)) -
            (k / ERC20(src).balanceOf(address(this)));

        ERC20(dst).transfer(msg.sender, out);

        uint kpost = token0.balanceOf(address(this)) * token1.balanceOf(address(this));

        assert(kpost >= k);
    }
}
```

AMM swap functional correctness (?)

Symbolic execution and model checking tools could not
prove/disprove the assertion

AMM swap functional correctness (?)
Fuzzing?

[illegible]

AMM swap functional correctness

Fuzzing with hevm

```
// swap allows the caller to exchange amt of src for dst at a price given
// by the constant product formula: x * y = k.
function swap(address src, address dst, uint amt) external {
    require(src != dst, "no self swap");
    require(src == address(token0) || src == address(token1), "src not in pair");
    require(dst == address(token0) || dst == address(token1), "dst not in pair");

    KPrev = token0.balanceOf(address(this)) * token1.balanceOf(address(this));

    ERC20(src).transferFrom(msg.sender, address(this), amt);

    uint out =
        ERC20(dst).balanceOf(address(this)) -
        (KPrev / ERC20(src).balanceOf(address(this)));

    ERC20(dst).transfer(msg.sender, out);

    KPost = token0.balanceOf(address(this)) * token1.balanceOf(address(this));
}

function invariant_k() public view {
    assert(KPost >= KPrev);
}

function kprev() public view returns (uint) {
    return KPrev;
}

function kpost() public view returns (uint) {
    return KPost;
}
```

```
contract TestAMM is DSTest {
    MintableERC20 token0;
    MintableERC20 token1;
    AMM pair;

    User user0;
    User user1;
    User user2;

    function targetContracts() public view returns (address[] memory) {
        address[] memory addrs = new address[](3);
        addrs[0] = address(user0);
        addrs[1] = address(user1);
        addrs[2] = address(user2);
        return addrs;
    }

    function setUp() public {
        token0 = new MintableERC20();
        token1 = new MintableERC20();
        pair = new AMM(address(token0), address(token1));

        user0 = new User(token0, token1, pair);
        user1 = new User(token0, token1, pair);
        user2 = new User(token0, token1, pair);

        token0.approve(address(pair), type(uint).max);
        token1.approve(address(pair), type(uint).max);

        uint amt = type(uint).max / 24;

        token0.mint(address(user0), amt);
        token0.mint(address(user1), amt);
        token0.mint(address(user2), amt);

        token1.mint(address(user0), amt);
        token1.mint(address(user1), amt);
        token1.mint(address(user2), amt);
    }

    function invariant_k() public view {
        pair.invariant_k();
    }
}
```

AMM swap functional correctness

Fuzzing with Echidna

```
contract TestAMM {
    MintableERC20 token0;
    MintableERC20 token1;
    AMM pair;

    User[3] users;

    constructor() {
        token0 = new MintableERC20();
        token1 = new MintableERC20();
        pair = new AMM(address(token0), address(token1));

        users[0] = new User(token0, token1, pair);
        users[1] = new User(token0, token1, pair);
        users[2] = new User(token0, token1, pair);

        token0.approve(address(pair), type(uint).max);
        token1.approve(address(pair), type(uint).max);

        uint amt = type(uint).max / 24;

        token0.mint(address(users[0]), amt);
        token0.mint(address(users[1]), amt);
        token0.mint(address(users[2]), amt);

        token1.mint(address(users[0]), amt);
        token1.mint(address(users[1]), amt);
        token1.mint(address(users[2]), amt);
    }

    function echidna_k() public returns (bool) {
        return pair.kpost() >= pair.kprev();
    }
}
```

AMM swap functional correctness

```
function swap(address src, address dst, uint amt) external {
    require(src != dst, "no self swap");
    require(src == address(token0) || src == address(token1), "src not in pair");
    require(dst == address(token0) || dst == address(token1), "dst not in pair");

    KPrev = token0.balanceOf(address(this)) * token1.balanceOf(address(this));

    ERC20(src).transferFrom(msg.sender, address(this), amt);

    uint out =
        ERC20(dst).balanceOf(address(this)) -
        (KPrev / ERC20(src).balanceOf(address(this)) + 1);

    ERC20(dst).transfer(msg.sender, out);

    KPost = token0.balanceOf(address(this)) * token1.balanceOf(address(this));
}
```

Model checkers still cannot prove correctness, but fuzzers could not find any other problems.

```
Fuzzing invariant
Running 1 tests for src/Amm.t.sol:TestAMM
[PASS] invariant_k() (runs: 100, depth: 20)
```

Echidna 1.7.1

```
Tests found: 1
Seed: 880666784831901063
Unique instructions: 4724
Unique codehashes: 4
Corpus size: 2
```

Tests

```
echidna_k: fuzzing (1280/50000)
```


Prove function correctness of deposit:
No tool could prove that the assertion is not reachable
automatically.

hevm results

```
Running Deposit contract  
checking postcondition...  
Q.E.D.  
Explored: 295 branches without assertion violations
```


SMTChecker says that the property does not hold

```
Warning: CHC: Assertion violation happens here.
Counterexample:
_totalSupply = 0
amount = 1
prevTotal = 115792089237316195423570985008687907853269984665640564039457584007913129639897

Transaction trace:
ERC777.constructor(115792089237316195423570985008687907853269984665640564039457584007913129639897)
State: _totalSupply = 115792089237316195423570985008687907853269984665640564039457584007913129639897
ERC777.burn(1)
    Context._msgSender() -- internal call
    ERC777._burn(7720, 1) -- internal call
        Context._msgSender() -- internal call
        ERC777._callTokensToSend(7720, 7720, 0, 1) -- internal call
            IERC777Sender(from).tokensToSend(operator, from, to, amount) -- untrusted external call, synthesized as:
                ERC777.burn(115792089237316195423570985008687907853269984665640564039457584007913129639897){ value: 14 } -- reentrant call
                    Context._msgSender(){ value: 14 } -- internal call
                    ERC777._burn(3565, 115792089237316195423570985008687907853269984665640564039457584007913129639897){ value: 14 } -- internal call
                        Context._msgSender(){ value: 14 } -- internal call
                        ERC777._callTokensToSend(3565, 3565, 0, 115792089237316195423570985008687907853269984665640564039457584007913129639897){ value: 14 } -- internal call
                            IERC777Sender(from).tokensToSend(operator, from, to, amount) -- untrusted external call
--> ERC777.sol:61:3:
|
|   |
61 |   |         assert(prevTotal == _totalSupply);
    |   |         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

- Remove strings (hard for counterexamples)

Property is indeed broken

```
dapp: Predeploying test library src/Address.sol:Address at 0x0DdeDfFe2789df61F5f76479464F568334414389
Running 1 tests for src/ERC777.t.sol:ERC777Test
[BAIL] test_totalSupply()

Failure: test_totalSupply
  src/ERC777.t.sol:ERC777Test
    - constructor
    - setUp()
      - create ERC777@0xCe71065D4017F316EC606Fe4422e11eB2c47c246 (src/ERC777.t.sol:29)
        ↳ 6856 bytes of code
      - create Rec@0x185a4dc360CE69bDCcE33b3784B0282f7961aea (src/ERC777.t.sol:30)
        ↳ 618 bytes of code
    - test_totalSupply()
      - call ERC777::transfer(address,uint256) (@0x185a4dc360CE69bDCcE33b3784B0282f7961aea, 100) (src/ERC777.t.sol:34)
        ↳ call ERC777Test::tokensToSend(address,address,address,uint256) (ERC777Test@0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84, ERC777Test@0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84, @0x185a4dc360CE69bDCcE33b3784B0282f7961aea, 100) (src/ERC777.sol:67)
          ↳ 0x
          - Sent(100) (src/ERC777.sol:323)
          - Transfer(100) (src/ERC777.sol:324)
          - call 0x185a4dc360CE69bDCcE33b3784B0282f7961aea::tokensReceived(ERC777Test@0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84, ERC777Test@0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84, @0x185a4dc360CE69bDCcE33b3784B0282f7961aea, 100) (src/ERC777.sol:84)
            ↳ call ERC777::burn(uint256)(100) (src/ERC777.t.sol:16)
              ↳ call 0x185a4dc360CE69bDCcE33b3784B0282f7961aea::tokensToSend(@0x185a4dc360CE69bDCcE33b3784B0282f7961aea, @0x185a4dc360CE69bDCcE33b3784B0282f7961aea, IERC777Sender@0x0000000000000000000000000000000000000000, 100) (src/ERC777.sol:67)
                ↳ 0x
                - Burned(100) (src/ERC777.sol:306)
                - Transfer(100) (src/ERC777.sol:307)
                ↳ 0x
                ↳ 0x
            - error Revert 0x4e487b7100000000000000000000000000000000000000000000000000000001 <source not found>
          - error Revert 0x4e487b7100000000000000000000000000000000000000000000000000000001 (src/ERC777.t.sol:34)
```


Are we safe now? Reentrancy guard blocks the tx before the assertion failure

```
dapp: Predeploying test library src/Address.sol:Address at 0x0DdeDfFe2789df61F5f76479464F568334414389
Running 1 tests for src/ERC777.t.sol:ERC777Test
[BAIL] test_totalSupply()

Failure: test_totalSupply
  src/ERC777.t.sol:ERC777Test
    - constructor
    - setUp()
      - create ERC777Mutex@0xCe71065D4017F316EC606Fe4422e11eB2c47c246 (src/ERC777.t.sol:40)
        ↳ 7124 bytes of code
      - create Rec@0x185a4dc360CE69bDCceE33b3784B0282f7961aea (src/ERC777.t.sol:41)
        ↳ 618 bytes of code
    - test_totalSupply()
      - call ERC777Mutex::transfer(address,uint256) (@0x185a4dc360CE69bDCceE33b3784B0282f7961aea, 100) (src/ERC777.t.sol:45)
        ↳ call ERC777Test::tokensToSend(address,address,address,uint256) (ERC777Test@0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84, ERC777Test@0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84, @0x185a4dc360CE69bDCceE33b3784B0282f7961aea, 100) (src/ERC777Mutex.sol:74)
          ↳ 0x
          ↳ Sent(100) (src/ERC777Mutex.sol:327)
          ↳ Transfer(100) (src/ERC777Mutex.sol:328)
          ↳ call 0x185a4dc360CE69bDCceE33b3784B0282f7961aea::tokensReceived(ERC777Test@0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84, ERC777Test@0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84, @0x185a4dc360CE69bDCceE33b3784B0282f7961aea, 100) (src/ERC777Mutex.sol:88)
            ↳ call ERC777Mutex::burn(uint256)(100) (src/ERC777.t.sol:21)
              ↳ error Revert 0x (src/ERC777Mutex.sol:42)
              ↳ error Revert 0x (src/ERC777.t.sol:21)
              ↳ error Revert 0x (src/ERC777Mutex.sol:88)
              ↳ error Revert 0x (src/ERC777.t.sol:45)
```

SMTChecker proves that now the assertions are safe

Contract invariants and external call properties for ERC777Mutex.sol:ERC777Mutex:

```
(!(<errorCode> >= 2) && (!lock || ((_totalSupply + ((- 1) * _totalSupply')) <= 0)) && (lock' || !lock) && (!lock || ((_totalSupply + ((- 1) * _totalSupply')) >= 0)))  
((!lock || !(<errorCode> >= 2)) && (!lock || ((_totalSupply + ((- 1) * _totalSupply')) <= 0)) && (lock' || !lock) && (!lock || ((_totalSupply + ((- 1) * _totalSupply')) >= 0)))
```

lock \implies ($_totalSupply = _totalSupply'$)

Conclusions

- ⚡ Automated FV tools can be quite powerful but...
- ⚡ No automated tool will do the job everytime
- ⚡ FV is still an expert domain
- ⚡ Specific tool knowledge is required to extract full potential
- ⚡ Playing with the tool is essential to get good results

Thank you!