runtime verification

The K Framework

A tool kit for language semantics and verification

PM Session

Smart Contract Verification with kontrol

Palina Tolmach

21st International Symposium on Automated Technology
for Verification and Analysis (ATVA 2023)
24 October 2023

PM Session Overview

runtime verification

- Intro to Blockchain, EVM, and Solidity
- Smart Contract Testing and Verification:
    - Unit Testing and Fuzzing with Foundry
    - Symbolic Execution with **kontrol**
- **kontrol** hands-on

Github repository for all materials

https://github.com/runtimeverification/k-tutorial-atva-2023

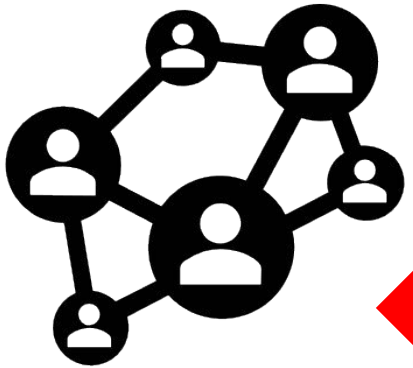
runtime verification

Intro to Blockhain

Blockchain

runtime verification

Managed by **peer-to-peer network**

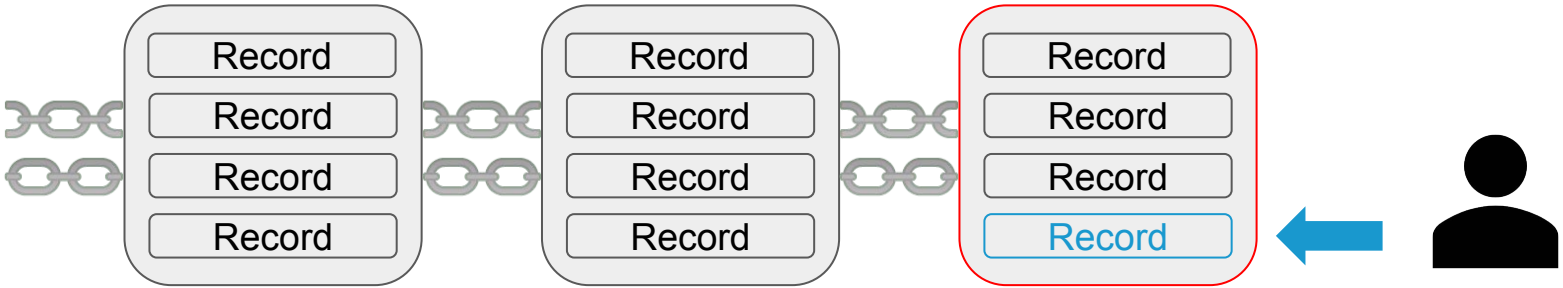


Which records to be included?

Consensus mechanism

Who can propose records?

Permissionless => Anyone

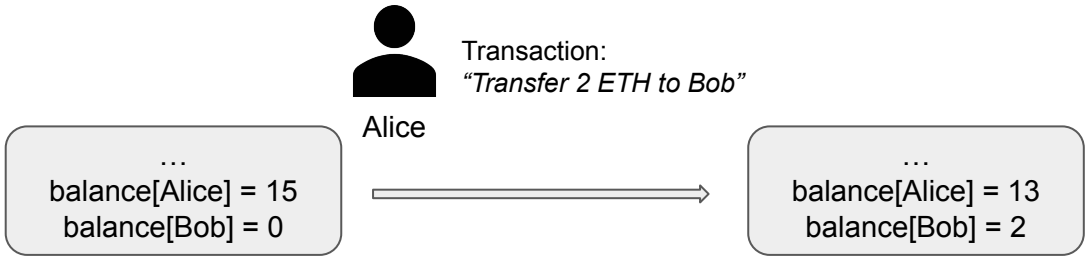


From Wiki: A **blockchain** is a **distributed** ledger with growing lists of records (blocks) that are securely linked together via cryptographic hashes.

Intro to Ethereum

Ethereum

- Transaction-based state machine
- Transactions are atomic
- ETH is native cryptocurrency



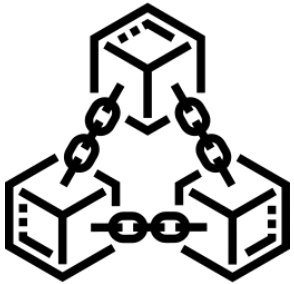
EVM — Source code & bytecode

```
pragma solidity ^0.8.13;

contract Counter {
    uint256 public number;

    function setNumber(uint256 newNumber) public {
        number = newNumber;
    }
}
```

```
608060405234801561000f575
15600e575f80fd5b506004361
942c14604e575b5f80fd5b603
f35b6054606d565b604051605
```



Solidity source code → Bytecode → EVM

```
❯ solc src/Counter.sol --bin
```

Record	Virtual Machine
Transaction	Universal Ledger (e.g., Bitcoin)
Transaction involving a smart contract	Universal Virtual Machine (e.g., Ethereum)

EVM

EVM — Opcode

runtime verification

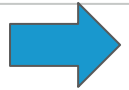
```
// SPDX-License-Identifier: UNLICENSED

pragma solidity ^0.8.13;

contract Counter {
    uint public count;

    function getCount() public view returns (uint) {
        return count;
    }
}
```

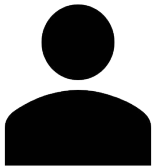
```
❯ solc src/Counter.sol --opcodes
```



```
1    PUSH1 0x80
2    PUSH1 0x40
3    MSTORE
4    CALLVALUE
5    DUP1
6    ISZERO
7    PUSH2 0xF
8    JUMPI
9    PUSH0
10   DUP1
11   REVERT
12   JUMPDEST
13   POP
14   PUSH1 0xD8
15   DUP1
16   PUSH2
17   0x1C
```

EVM — Transaction & State Update

runtime verification

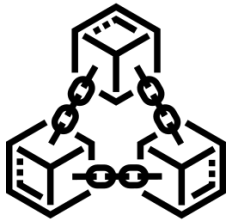


3

```
pragma solidity ^0.8.13;

contract Counter {
    uint256 public number;

    function setNumber(uint256 newNumber) public {
        number = newNumber;
    }
}
```



number: 3	…

runtime verification

Intro to Solidity Development

Solidity

runtime verification

- Statically-typed, curly-bracket language, inspired by C++, JS, Python
- Contracts are like objects in OOP containing state variables, functions
- Has contract-specific features, e.g.,
  - custom global (blockchain) variables
  - require (guard) clauses
  - etc.
- Latest version is 0.8.21

```
pragma solidity ^0.8.13;

contract Counter {
    uint256 public number;
    function setNumber(uint256 newNumber, address sender) public {
        require(sender == msg.sender);
        number = newNumber;
        assert(number == 3);
    }
}
```

runtime verification

Smart Contract Testing and Verification

Permissionless => Issues

runtime verification

Permissionless

- anyone can write a smart contract and deploy it to the network
  - anyone can inspect deployed smart contracts
- anyone can issue a transaction / interact with a smart contract

⚠️ Security issues ⚠️

Security Incidents

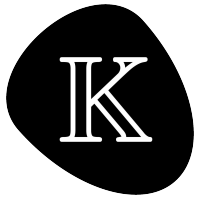
rekt

1. **Ronin Network - REKT** *Unaudited*
   $624,000,000 | 03/23/2022
2. **Poly Network - REKT** *Unaudited*
   $611,000,000 | 08/10/2021
3. **BNB Bridge - REKT** *Unaudited*
   $586,000,000 | 10/06/2022
4. **SBF - MASK OFF** *N/A*
   $477,000,000 | 11/12/22
5. **Wormhole - REKT** *Neodyme*
   $326,000,000 | 02/02/2022
6. **Mixin Network - REKT** *N/A*
   $200,000,000 | 09/23/2023
7. **Euler Finance - REKT** *Sherlock*
   $197,000,000 | 03/13/2023
8. **BitMart - REKT** *N/A*
   $196,000,000 | 12/04/2021
9. **Nomad Bridge - REKT** *N/A*
   $190,000,000 | 08/01/2022
10. **Beanstalk - REKT** *Unaudited*
    $181,000,000 | 04/17/2022
11. **Wintermute - REKT 2** *N/A*
    $162,300,000 | 09/20/2022
12. **Compound - REKT** *Unaudited*
    $147,000,000 | 09/29/2021
13. **Vulcan Forged - REKT** *Unaudited*
    $140,000,000 | 12/13/2021
14. **Cream Finance - REKT 2** *Unaudited*
    $130,000,000 | 10/27/2021
15. **Multichain - REKT 2** *N/A*
    $126,300,000 | 07/06/2023
16. **BonqDAO - REKT** *Out of scope*
    $120,000,000 | 02/01/2023
17. **Badger - REKT** *Unaudited*
    $120,000,000 | 12/02/2021
18. **Mango Markets - REKT** *Out of Scope*
    $115,000,000 | 10/11/2022
19. **Atomic Wallet - REKT** *Unaudited*
    $100,000,000 | 06/02/2023
20. **Harmony Bridge - REKT** *N/A*
    $100,000,000 | 06/23/2022

runtime verification

Property Testing Tools

runtime verification

Unit testing	 Foundry	 Hardhat
Fuzz testing	 Foundry	 ERCX
Formal verification		 kontrol

The Foundry Toolkit

runtime verification

- Very **easy** to use - write property tests and test.
- Supports **fuzzing** over parametric tests at the Solidity level.
- Blazing **fast** execution, providing users with immediate feedback.
- There are no false positives[1], but it can produce false negatives[2].

[1]Tests that are supposed to pass will pass.

[2]Some tests that are supposed to fail may pass because the fuzzing test cannot find counterexamples within the limited number of runs.

Foundry Cheat codes

runtime verification

```
54      function startPrank(address) external;
55      // Sets the *next* call's msg.sender to be the input address, and the tx.origin to be the second input
56      function prank(address,address) external;
57      // Sets all subsequent calls' msg.sender to be the input address until `stopPrank` is called, and the tx.origin to be the second input
58      function startPrank(address,address) external;
59      // Resets subsequent calls' msg.sender to be `address(this)`
60      function stopPrank() external;
61      // Sets an address' balance, (who, newBalance)
62      function deal(address, uint256) external;
63      // Sets an address' code, (who, newCode)
64      function etch(address, bytes calldata) external;
65      // Expects an error on next call
66      function expectRevert(bytes calldata) external;
67      function expectRevert(bytes4) external;
68      function expectRevert() external;
```

- A Solidity Interface that contains function signatures
- Give developers the ability to alter the state of the EVM from their own Solidity tests

Foundry Fuzz Testing

runtime verification

```
function testBalanceOf(address addr, uint256 amount) public {
    ERC20 erc20 = new ERC20("Bucharest Workshop Token", "BWT");
    bytes32 storageLocation = getStorageLocationForKey(addr, BALANCES_STORAGE_INDEX);
    vm.assume(uint256(vm.load(address(erc20), storageLocation)) == amount);
    uint256 balance = erc20.balanceOf(addr);
    assertEq(balance, amount);
}
```

- Foundry would fuzz the values of the function parameters.

Foundry Fuzz Testing Output

runtime verification

```
anvacaru@desktop foundry-demo$ FOUNDRY_FUZZ_RUNS=300 forge test
[⠒] Compiling...
No files changed, compilation skipped

Running 8 tests for test/ERC20.t.sol:ERC20Test
[PASS] testBalanceOf(address,uint256) (runs: 300, μ: 510340, ~: 510340)
[PASS] testName() (gas: 508343)
[PASS] testSymbol() (gas: 508387)
[PASS] testTotalSupply(uint256) (runs: 300, μ: 509813, ~: 509813)
[PASS] testTransferFailure_0(address,uint256) (runs: 300, μ: 510167, ~: 510167)
[PASS] testTransferFailure_1(uint256) (runs: 300, μ: 509723, ~: 509723)
[PASS] testTransferFailure_2() (gas: 513545)
[FAIL. Reason: The `vm.assume` cheatcode rejected too many inputs (65536 allowed)]
,address,uint256,uint256,uint256) (runs: 9, μ: 516918, ~: 516918)
Test result: FAILED. 7 passed; 1 failed; finished in 3.59s
```

But recall that …

Some tests that are supposed to fail may pass because the fuzzing test cannot find counterexamples within the limited number of runs.

runtime verification

Foundry hands-on

Foundry Installation & Project Initialization

runtime verification

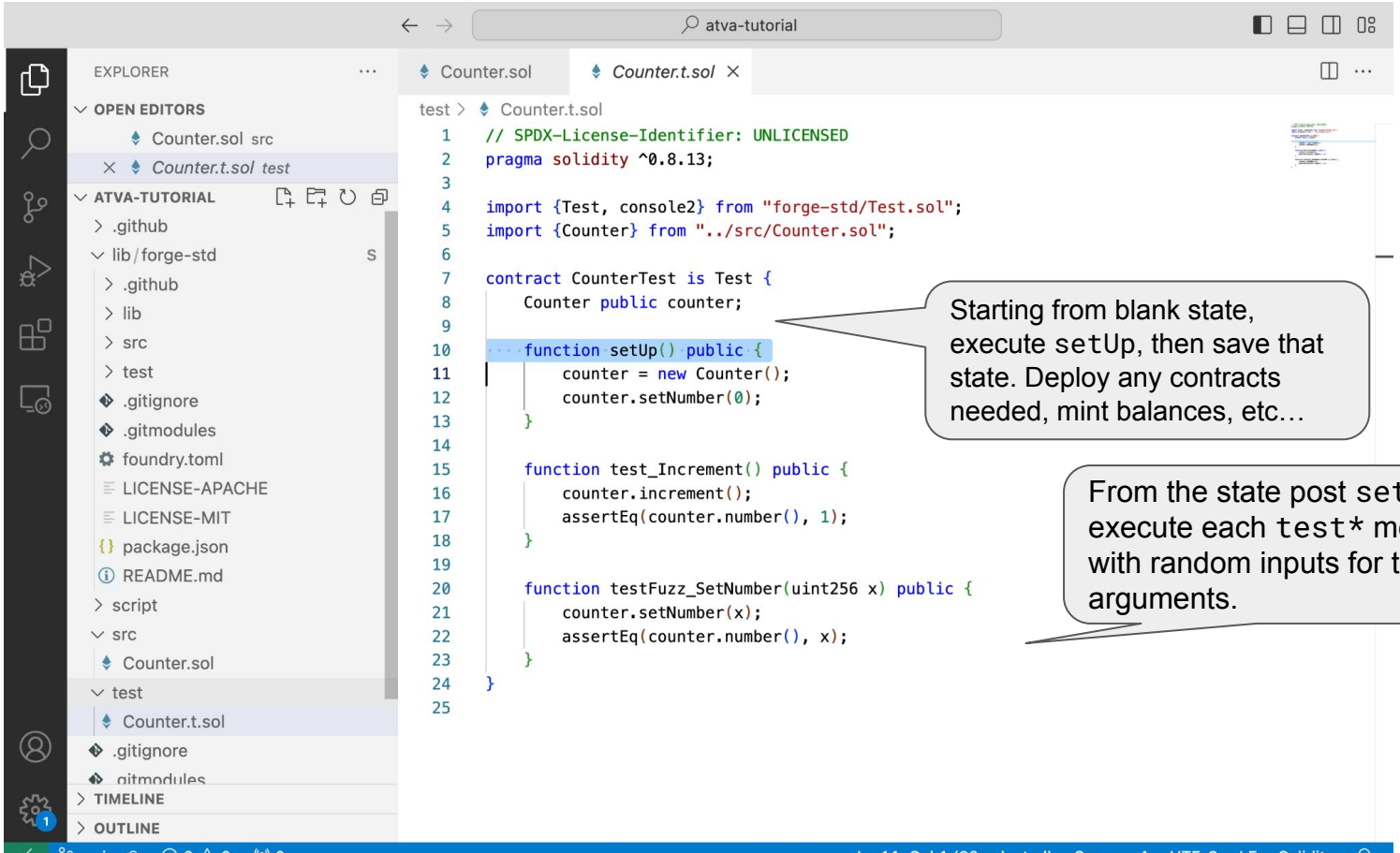
- `curl -L https://foundry.paradigm.xyz | bash`
- `foundryup`
- *…depending on your system, additional steps might be needed…*

// creating & initializing the project

- `mkdir foundry-project && cd foundry-project`
- `forge init --no-commit`

Foundry Project Structure

runtime verification



Testing & Fuzzing w/Foundry (1)

runtime verification

- Run `forge test`

```
❯ forge test
[⠒] Compiling...
[⠒] Compiling 1 files with 0.8.21
[⠒] Solc 0.8.21 finished in 818.52ms
Compiler run successful!

Running 2 tests for test/Counter.t.sol:CounterTest
[PASS] testFuzz_SetNumber(uint256) (runs: 256, μ: 27864, ~: 28409)
[PASS] test_Increment() (gas: 28379)
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 11.14ms

Ran 1 test suites: 2 tests passed, 0 failed, 0 skipped (2 total tests)
```

- Now, add the following test to `Counter.t.sol`:

```
function test_failure(uint256 x) public {
    if (x == 4) {
        assert(false);
    }
}
```

- Add more digits to the constant (e.g., `4` -> `421`)

Foundry Project Configuration

runtime verification

foundry.toml M

foundry.toml

```
1   [profile.default]
2   src = "src"
3   out = "out"
4   libs = ["lib"]
5
6   [fuzz]
7   runs = 65536
8
```

PROBLEMS OUTPUT TERMINAL ··· zsh

```
Running 3 tests for test/Counter.t.sol:CounterTest
[PASS] testFuzz_SetNumber(uint256) (runs: 65536, μ: 27583, ~: 28387)
[PASS] test_Increment() (gas: 28357)
[FAIL. Reason: Assertion violated Counterexample: calldata=0x0xf4e6432e00000000000000
0000000000000000000000000000000000000000000001a5, args=[421]] test_failure(uint256)
 (runs: 454, μ: 312, ~: 312)
Test result: FAILED. 2 passed; 1 failed; 0 skipped; finished in 1.38s

Ran 1 test suites: 2 tests passed, 1 failed, 0 skipped (3 total tests)
```

More config options:
https://github.com/foundry-rs/foundry/blob/master/crates/config/README.md

Symbolic execution using kontrol

What is KEVM?

runtime verification

- KEVM = Formal semantics of the Ethereum Virtual Machine in K Framework.
  - Passes same conformance test-suite as other clients.
  - Enables symbolic execution (and thus verification) of EVM bytecode.
  - Online: https://jellopaper.org or https://github.com/runtimeverification/evm-semantics

- Defined using a configuration and transition rules.
- Used by Runtime Verification in formal engagements.
- Large-scale proving with K and ACT (from Multi-Collateral Dai system - 1011 proofs)

KEVM Former Approach

runtime verification

```
6     // balanceOf
7     claim
8       <kevm>
9       <k> #execute => #halt </k>
10
11      <ethereum>
12        <output> _ => #buf(32, BAL) </output>
13        <statusCode> _ => EVMC_SUCCESS </statusCode>
14
15        <id> ACCT_ID </id> // contract owner
16        <caller> CALLER_ID </caller> // who called this contract; in the begining, origin // msg.sende
17
18        <callData> #abiCallData("balanceOf", #address(OWNER)) </callData>
19
20        <account>
21          <acctID> ACCT_ID </acctID>
22          <balance> _ </balance>
23          <code> #parseByteStack("0x6060604052600436106100d576000357c0100000000000000000000000000000000
24          <storage> #hashedLocation("Solidity", 1, OWNER) |-> BAL
25                    _:Map
26          </storage>
27        </account>
100     requires 0 <=Int ACCT_ID    andBool ACCT_ID    <Int (2 ^Int 160)
101      andBool 0 <=Int CALLER_ID  andBool CALLER_ID  <Int (2 ^Int 160)
102      andBool 0 <=Int ORIGIN_ID  andBool ORIGIN_ID  <Int (2 ^Int 160)
103      andBool 0 <=Int CALL_DEPTH andBool CALL_DEPTH <Int 1024
104      andBool 0 <=Int OWNER      andBool OWNER      <Int (2 ^Int 160)
105   andBool 0 <=Int BAL        andBool BAL        <Int (2 ^Int 256)
```

```
contract ERC20 {

    mapping(address => uint256) private _balances;

    function balanceOf(address account) external view returns (uint256) {
        return _balances[account];
    }
```

Provide formal verification
BUT
difficult to write proofs

KEVM Former Approach

runtime verification

```
requires "../verification.k"

module BALANCEOF-SPEC
  imports VERIFICATION

  // balanceOf
  claim
    <kevm>
    <k> #execute => #halt </k>
    <exit-code> 1 </exit-code>
    <mode> NORMAL </mode>
    <schedule> ISTANBUL </schedule>

    <ethereum>
      <evm>
        <output> _ => #buf(32, BAL) </output>
        <statusCode> _ => EVMC_SUCCESS </statusCode>
        <callStack> _ </callStack>
        <interimStates> _ </interimStates>
        <touchedAccounts> _ => ?_ </touchedAccounts>

        <callState>
          <program> #parseByteStack("0x60606040526004361061006d576000357c0100000000000000000000000000000000000000000000000000000000900000
          <jumpDests> #computeValidJumpDests(#parseByteStack("0x60606040526004361061006d576000357c0100000000000000000000000000000000000000000000000000000000900000

          <id> ACCT_ID </id> // contract owner
          <caller> CALLER_ID </caller> // who called this contract; in the begining, origin // msg.sender

          <callData> #abiCallData("balanceOf", #address(OWNER)) </callData>

          <callValue> 0 </callValue>
          <wordStack> .WordStack => ?_ </wordStack>
          <localMem> .Bytes => ?_ </localMem>
          <pc> 0 => ?_ </pc>
          <gas> #gas(_VGAS) => ?_ </gas>
          <memoryUsed> 0 => ?_ </memoryUsed>
          <callGas> _ => ?_ </callGas>

          <static> false </static> // NOTE: non-static call
          <callDepth> CALL_DEPTH </callDepth>
        </callState>

        <substate>
          <selfDestruct> _ </selfDestruct>
          <log> _ </log>
          <refund> _ </refund> // TODO: more detail
          <accessedAccounts> _ => ?_ </accessedAccounts>
          <accessedStorage> _ => ?_ </accessedStorage>
        </substate>

        <gasPrice> _ </gasPrice>
        <origin> ORIGIN_ID </origin> // who fires tx

        <blockhashes> _ </blockhashes>
        <block>
```

```
        <block>
          <previousHash> _ </previousHash>
          <ommersHash> _ </ommersHash>
          <coinbase> _ </coinbase>
          <stateRoot> _ </stateRoot>
          <transactionsRoot> _ </transactionsRoot>
          <receiptsRoot> _ </receiptsRoot>
          <logsBloom> _ </logsBloom>
          <difficulty> _ </difficulty>
          <number> _ </number>
          <gasLimit> _ </gasLimit>
          <gasUsed> _ </gasUsed>
          <timestamp> _ </timestamp>
          <extraData> _ </extraData>
          <mixHash> _ </mixHash>
          <blockNonce> _ </blockNonce>
          <baseFee> _ </baseFee>
          <ommerBlockHeaders> _ </ommerBlockHeaders>
        </block>
      </evm>

      <network>
        <chainID> _ </chainID>

        <activeAccounts> SetItem(ACCT_ID) _:Set </activeAccounts>

        <accounts>
          <account>
            <acctID> ACCT_ID </acctID>
            <balance> _ </balance>
            <code> #parseByteStack("0x60606040526004361061006d576000357c0100000000000000000000000000000000000000000000000000000000900000
            <storage> #hashedLocation("Solidity", 1, OWNER) |-> BAL
_:Map </storage>
            <origStorage> _ </origStorage>
            <nonce> _ </nonce>
          </account>
        // ... // TODO: fix
        </accounts>

        <txOrder> _ </txOrder>
        <txPending> _ </txPending>
        <messages> _ </messages>
      </network>
    </ethereum>
    </kevm>
    requires 0 <=Int ACCT_ID    andBool ACCT_ID    <Int (2 ^Int 160)
     andBool 0 <=Int CALLER_ID  andBool CALLER_ID  <Int (2 ^Int 160)
     andBool 0 <=Int ORIGIN_ID  andBool ORIGIN_ID  <Int (2 ^Int 160)
     andBool 0 <=Int CALL_DEPTH andBool CALL_DEPTH <Int 1024
     andBool 0 <=Int OWNER      andBool OWNER      <Int (2 ^Int 160)
andBool 0 <=Int BAL       andBool BAL       <Int (2 ^Int 256)


endmodule
```

Aiming for Formal Verification

runtime verification

- Hoare Triples - `(∀ vars){pre-conditions} code {post-conditions}`
- We can use Foundry and KEVM to define Hoare Triples in Solidity parametric tests.

```
function testProperty(vars) external {
   assume pre;
   code;
   assert post;
}
```

- Using the symbolic execution capabilities of KEVM, it will formally verify the specifications.

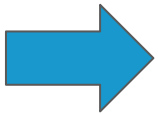
KEVM former approach might be too challenging

Property test function from Foundry

(Easy to write)

Symbolic execution of KEVM

(Formal verification)



kontrol

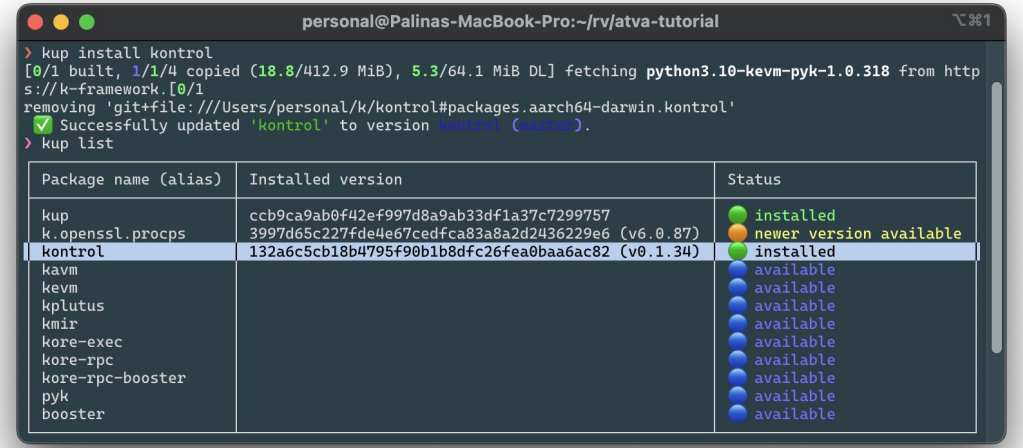
runtime verification

kontrol hands-on

kontrol Installation

runtime verification

- `bash <(curl https://kframework.org/install)`
- **`kup install kontrol`**
- `kup list`



kontrol hands-on: Example #1 (Counter)

runtime verification

- Let's make the `Counter` contract more interesting:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

error CoffeeBreak();

contract Counter {
   uint256 public number;

   function setNumber(uint256 newNumber, bool inLuck) public {
       number = newNumber;
       if (newNumber == 0xC0FFEE && inLuck == true) {
           revert CoffeeBreak();
       }
   }

   function increment() public {
        number++;
    }
}
```

kontrol hands-on: Example #1

runtime verification

- And `testSetNumber` — more challenging to verify

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "../src/Counter.sol";

contract CounterTest is Test {
   Counter public counter;

   function setUp() public {
       counter = new Counter();
       counter.setNumber(0, false);
   }

   function testIncrement() public {
        counter.increment();
        assertEq(counter.number(), 1);
    }

   function testSetNumber(uint256 x, bool inLuck) public {
       counter.setNumber(x, inLuck);
       assertEq(counter.number(), x);
   }
}
```

kontrol hands-on: Example 1

runtime verification

- Compile:
  - `kontrol build --verbose`
- Verify:
  - `kontrol prove --test CounterTest.testSetNumber \`
    `--use-booster \`
    `- counterexample-information`
- Examine:
  - `kontrol view-kcfg --test CounterTest.testSetNumber`
  - `kontrol list`

kontrol hands-on: Example 2

runtime verification

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

error CoffeeBreak();

contract Counter {
    uint256 public number;

    bool public isActive;

    function activate() public {
        isActive = true;
    }

    function setNumber(uint256 newNumber, bool inLuck) public {
        number = newNumber;
        if (newNumber == 0xC0FFEE && inLuck == true && isActive == true) {
            revert CoffeeBreak();
        }
    }

    function increment() public {
        number++;
    }
}
```

kontrol hands-on: Kontrol Cheatcodes

runtime verification

- `forge install runtimeverification/kontrol-cheatcodes --no-commit`

```
interface KontrolCheatsBase {
    // Expects a call using the CALL opcode to an address with the specified calldata.
    function expectRegularCall(address,bytes calldata) external;
    // Expects a call using the CALL opcode to an address with the specified msg.value and calldata.
    function expectRegularCall(address,uint256,bytes calldata) external;
    // Expects a static call to an address with the specified calldata.
    function expectStaticCall(address,bytes calldata) external;
    // Expects a delegate call to an address with the specified calldata.
    function expectDelegateCall(address,bytes calldata) external;
    // Expects that no contract calls are made after invoking the cheatcode.
    function expectNoCall() external;
    // Expects the given address to deploy a new contract, using the CREATE opcode, with the specified value and bytecode.
    function expectCreate(address,uint256,bytes calldata) external;
    // Expects the given address to deploy a new contract, using the CREATE2 opcode, with the specified value and bytecode (appended with a bytes32 salt).
    function expectCreate2(address,uint256,bytes calldata) external;
    // Makes the storage of the given address completely symbolic.
    function symbolicStorage(address) external;
    // Adds an address to the whitelist.
    function allowCallsToAddress(address) external;
    // Adds an address and a storage slot to the whitelist.
    function allowChangesToStorage(address,uint256) external;
    // Sets the remaining gas to an infinite value.
    function infiniteGas() external;
    // Sets the current <gas> cell to the supplied amount.
    function setGas(uint256) external;
    // Returns a symbolic unsigned integer
    function freshUInt(uint8) external returns (uint256);
    // Returns a symbolic boolean value
    function freshBool() external returns (uint256);
}

abstract contract KontrolCheats {
    KontrolCheatsBase public constant kevm = KontrolCheatsBase(address(uint160(uint256(keccak256("hevm cheat code")))));

    // Checks if an address matches one of the built-in addresses.
    function notBuiltinAddress(address addr) internal pure returns (bool) {
        return (addr != address(6453264744265472033134100691539059085253624343 49) &&
                addr != address(7288155633859770404529437778790614277562773065 18));
    }
}
```

kontrol hands-on: Example 2

runtime verification

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "../src/Counter.sol";
import "kontrol-cheatcodes/KontrolCheats.sol";

contract CounterTest is Test, KontrolCheats {
    Counter public counter;

    function setUp() public {
        counter = new Counter();
    //    counter.activate();
        counter.setNumber(0, false);
    }

    function testIncrement() public {
        counter.increment();
        assertEq(counter.number(), 1);
    }

    function testSetNumber(uint256 x, bool inLuck) public {
        kevm.symbolicStorage(address(counter));
        counter.setNumber(x, inLuck);
        assertEq(counter.number(), x);
    }
}
```

kontrol hands-on: Example 2

runtime verification

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "../src/Counter.sol";
import "kontrol-cheatcodes/KontrolCheats.sol";

contract CounterTest is Test, KontrolCheats {
    Counter public counter;

    function setUp() public {
        counter = new Counter();
     //    counter.activate();
        counter.setNumber(0, false);
    }

    function testIncrement() public {
         counter.increment();
         assertEq(counter.number(), 1);
     }

    function testSetNumber(uint256 x, bool inLuck) public {
        kevm.symbolicStorage(address(counter));
        counter.setNumber(x, inLuck);
        assertEq(counter.number(), x);
    }
}
```

```
> kontrol prove --use-booster
```

```
PROOF FAILED: ('CounterTest.testSetNumber(uint256,bool)', 0)
1 Failure nodes. (0 pending and 1 failing)

Failing nodes:

  Node id: 64
  Failure reason:
    Implication check failed, the following is the remaining implication:
    ( ( { VV0_x_114b9705:Int #Equals 12648430 }
    #And ( { true #Equals VV1_inLuck_114b9705:Int ==Int 1 ==Bool true }
    #And ( { true #Equals #rangeBool ( VV1_inLuck_114b9705:Int ) }
    #And { true #Equals ( notBool ( maxUInt8 &Int #lookup ( ?STORAGE:Map , 1 ) ) ==Int 0 ) ==Bool true } ) ) ) #Implie
s { true #Equals foundry_success ( ... statusCode: EVMC_REVERT , failed: #lookup ( .Map , 463080223264950070279727286 7
7917914892729792999299745830475596687180801507328 ) , revertExpected: false , opcodeExpected: false , recordEventExpec
ted: false , eventExpected: false ) } )
  Path condition:
    ( { true #Equals ( notBool ( notBool VV0_x_114b9705:Int ==Int 12648430 ) ) }
#And ( { true #Equals ( notBool ( notBool VV1_inLuck_114b9705:Int ==Int 1 ==Bool true ) ) }
#And { true #Equals ( notBool ( notBool ( notBool ( maxUInt8 &Int #lookup ( ?STORAGE:Map , 1 ) ) ==Int 0 ) ==Bool true
```

kontrol hands-on: Example 3 (Solady)

runtime verification

```
contract Solady {
    /// @dev The scalar of ETH and most ERC20s.
    uint256 internal constant WAD = 1e18;

    /// @dev Equivalent to `(x * y) / WAD` rounded down.
    function mulWad(uint256 x, uint256 y) public pure returns (uint256 z) {
        /// @solidity memory-safe-assembly
        assembly {
            // Equivalent to `require(y == 0 || x <= type(uint256).max / y)`.
            if mul(y, gt(x, div(not(0), y))) {
                mstore(0x00, 0xbac65e5b) // `MulWadFailed()`.
                revert(0x1c, 0x04)
            }
            z := div(mul(x, y), WAD)
        }
    }
}
```

https://github.com/Vectorized/solady

kontrol hands-on: Example 3 (Solady)

runtime verification

```
function testMulWad(uint256 x, uint256 y) public {
    if(y == 0 || x <= type(uint256).max / y) {
        uint256 zSpec = (x * y) / WAD;
        uint256 zImpl = solady.mulWad(x, y);
        assertEq(zImpl, zSpec);
    } else {
        vm.expectRevert();
        solady.mulWad(x, y);
    }
}
```

```
> kontrol build
> kontrol prove --test SoladyTest.testMulWad --use-booster
> kontrol view-kcfg --test SoladyTest.testMulWad
```

kontrol hands-on: Example 3 (Solady)

runtime verification

- Adding lemmas:

lemmas.k

```
1   requires "evm.md"
2   requires "foundry.md"
3
4   module DEMO-LEMMAS
5       imports BOOL
6       imports FOUNDRY
7       imports INFINITE-GAS
8       imports INT-SYMBOLIC
9
10      rule bool2Word ( X ) => 1 requires X         [simplification]
11      rule bool2Word ( X ) => 0 requires notBool X [simplification]
12  endmodule
```

```
> kontrol build --rekompile --require ./lemmas.k --module-import SoladyTest:DEMO-LEMMAS
> kontrol prove --test SoladyTest.testMulWad --use-booster--reinit
```

kontrol hands-on: Example 4 (Loops)

runtime verification

```
contract LoopsTest is Test {
    function sum_N(uint n) public returns (uint) {
        vm.assume(n <= 51816696836262767);

        uint s = 0;
        while (0 < n) {
            s = s + n;
            n = n - 1;
        }
        return s;
    }

    function test_sum_10() public returns (uint) {
        return sum_N(10);
    }
}
```

kontrol hands-on: Example 4 (Loops)

runtime verification

```
> kontrol prove --test LoopsTest.test_sum_10 --use-booster
> kontrol prove --test LoopsTest.sum_N

// bound loop iterations
> kontrol prove --test LoopsTest.sum_N --bmc-depth 3

// add loop invariant
> kontrol build --rekompile --require ./invariant_lemmas.k --module-import
LoopsTest:SUM-TO-N-INVARIANT
> kontrol prove --test LoopsTest.sum_N --bmc-depth 3
```

kontrol hands-on: Example 5 (`wmul`)

runtime verification

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";

contract Examples is Test {
    uint256 constant MAX_INT = (2 ** 256) - 1;
    uint constant WAD = 10 ** 18;

    function wmul(uint x, uint y) internal pure returns (uint z) {
        z = (x * y) / WAD;
    }

    function test_wmul_increasing(uint a, uint b) public {
        uint c = wmul(a, b);
        // overflow check
        assertTrue(a < c && b < c);
    }
}
```

Step-by-step tutorial

runtime verification

Github repository for all materials

https://github.com/runtimeverification/k-tutorial-atva-2023

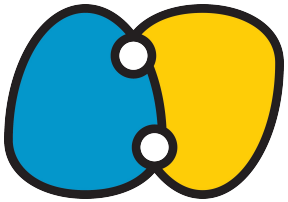
More on kontrol

runtime verification

Find out more at

https://docs.runtimeverification.com/kontrol/overview/readme

runtime verification



Questions?

https://runtimeverification.com/

@rv_inc

https://discord.com/invite/CurfmXNtbN

contact@runtimeverification.com