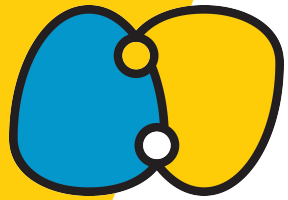
runtime verification



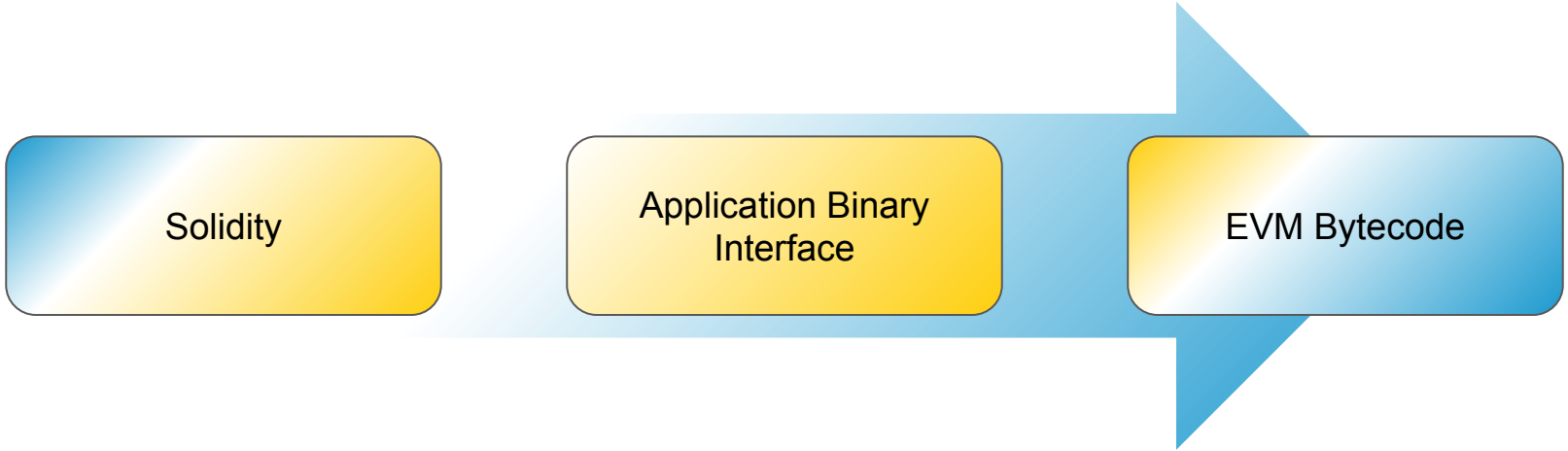
Smart contract tooling using KEVM

Andrei Văcaru

Runtime Verification Inc.

Getting up to speed

runtime verification



Tooling

- Provided a local node with an EVM implementation to act as a sandbox for developers.
- Unit testing available in both JavaScript and Solidity.
- Widely adopted with active and supportive communities.

Foundry cheat codes

runtime verification

- A Solidity Interface that contains function signatures.
- Do not have an implementation at the Solidity level.
- Give developers the ability to alter the state of the EVM from their own Solidity tests.

Cheat codes in Solidity

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;
```

Cheat codes in Solidity

runtime verification

```
486  library stdStorage {
487      event SlotFound(address who, bytes4 fsig, bytes32 keysHash, uint slot);
488      event WARNING_UninitedSlot(address who, uint slot);
489
490      uint256 private constant UINT256_MAX =
         115792089237316195423570985008687907853269984665640564039457584007913129639935;
491      int256 private constant INT256_MAX =
         57896044618658097711785492504343953926634992332820282019728792003956564819967;
492
493      Vm private constant vm_std_store = Vm(address(uint160(uint256(keccak256('hevm cheat
         code')))));
```

Cheat codes in KEVM

runtime verification

```
    rule [foundry.call]:
         <k> (#checkCall _ _
          ~> #call _ CHEAT_ADDR _ _ _ ARGS _
          ~> #return RETSTART RETWIDTH )
          => #call_foundry #asWord(#range(ARGS, 0, 4)) #range(ARGS, 4, lengthBytes(ARGS) -Int 4)
          ~> #return_foundry RETSTART RETWIDTH
         ...
         </k>
         <output> _ => .Bytes </output>
    requires CHEAT_ADDR ==Int #address(FoundryCheat)
    [priority(40)]

    rule [foundry.call.deal]:
         <k> #call_foundry SELECTOR ARGS => #loadAccount #asWord(#range(ARGS, 0, 32)) ~>
           #setBalance #asWord(#range(ARGS, 0, 32)) #asWord(#range(ARGS, 32, 32)) ... </k>
      requires SELECTOR ==Int selector ( "deal(address,uint256)" )
```

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

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

Failing tests:
Encountered 1 failing test in test/ERC20.t.sol:ERC20Test
[FAIL. Reason: The `vm.assume` cheatcode rejected too many inputs (65536 allowed)]
```

Foundry Fuzz Testing

runtime verification

```
function testTransferSuccess(address alice, address bob, uint256 balanceA, uint256 balanceB,
uint256 amount) public {
    bytes32 storageIndexA = getStorageLocationForKey(alice, BALANCES_STORAGE_INDEX);
    bytes32 storageIndexB = getStorageLocationForKey(bob  , BALANCES_STORAGE_INDEX);
    ERC20 erc20 = new ERC20("Bucharest Workshop Token", "BWT");
    vm.assume(uint256(vm.load(address(erc20), storageIndexA)) == balanceA);
    vm.assume(uint256(vm.load(address(erc20), storageIndexB)) == balanceB);
    vm.assume(amount <= balanceA);
    vm.assume(alice != address(0));
    vm.assume(bob != address(0));
    vm.startPrank(address(alice));      You, 4 minutes ago • Uncommitted changes
    erc20.transfer(alice, amount);
    assert(uint256(vm.load(address(erc20), storageIndexA)) == balanceA - amount);
    assert(uint256(vm.load(address(erc20), storageIndexB)) == balanceB + amount);
}
```

KEVM

runtime verification

- Online: https://jellopaper.org
- GitHub: https://github.com/runtimeverification/evm-semantics
- K semantics of the Ethereum Virtual Machine.
  - Passes same conformance test-suite as other clients.
  - Enables symbolic execution (and thus verification) of EVM bytecode.
- Example standalone K proof (`transfer` function of an ERC20)
- Large-scale proving with K and ACT (from Multi-Collateral Dai system - 1011 proofs)

ERC20 claim in KEVM

runtime verification

```
6   // balanceOf
7   claim
8       <kevm>
9       <k> #execute => #halt </k>
10      <exit-code> 1 </exit-code>
11      <mode> NORMAL </mode>
12      <schedule> ISTANBUL </schedule>
13
14      <ethereum>
15          <evm>
16              <output> _ => #buf(32, BAL) </output>
17              <statusCode> _ => EVMC_SUCCESS </statusCode>
18              <callStack> _ </callStack>
19              <interimStates> _ </interimStates>
20              <touchedAccounts> _ => ?_ </touchedAccounts>
21
22              <callState>
23                  <program> #parseByteStack("0x6060604052600436106100d576000357c010000000000000000
24                  <jumpDests> #computeValidJumpDests(#parseByteStack("0x6060604052600436106100d5760
```

ERC20 claim in KEVM

runtime verification

```
29            <callData> #abiCallData("balanceOf", #address(OWNER)) </callData>
30
31            <callValue> 0 </callValue>
32            <wordStack> .WordStack => ?_ </wordStack>
33            <localMem> .Bytes => ?_ </localMem>
34            <pc> 0 => ?_ </pc>
35            <gas> #gas(_VGAS) => ?_ </gas>
36            <memoryUsed> 0 => ?_ </memoryUsed>
37            <callGas> _ => ?_ </callGas>
38
39            <static> false </static> // NOTE: non-static call
40            <callDepth> CALL_DEPTH </callDepth>
41        </callState>
42
43        <substate>
44            <selfDestruct> _ </selfDestruct>
45            <log> _ </log>
```

ERC20 claim in KEVM

runtime verification

```
29            <callData> #abiCallData("balanceOf", #address(OWNER)) </callData>
30
31            <callValue> 0 </callValue>
32            <wordStack> .WordStack => ?_ </wordStack>
33            <localMem> .Bytes => ?_ </localMem>
34            <pc> 0 => ?_ </pc>
35            <gas> #gas(_VGAS) => ?_ </gas>
36            <memoryUsed> 0 => ?_ </memoryUsed>
37            <callGas> _ => ?_ </callGas>
38
39            <static> false </static> // NOTE: non-static call
40            <callDepth> CALL_DEPTH </callDepth>
41          </callState>
42
43          <substate>
44            <selfDestruct> _ </selfDestruct>
45            <log> _ </log>
```

calldata

Infinite gas

ERC20 claim in KEVM

runtime verification

```
81          <accounts>
82            <account>
83              <acctID> ACCT_ID </acctID>
84              <balance> _ </balance>
85              <code> #parseByteStack("0x6060604052600436106100d576000357c0100000000000000000
86              <storage> #hashedLocation("Solidity", 1, OWNER) |-> BAL
87  _:Map </storage>
88              <origStorage> _ </origStorage>
89              <nonce> _ </nonce>
90            </account>

96        </network>
97      </ethereum>
98      </kevm>
99      requires 0 <=Int ACCT_ID    andBool ACCT_ID    <Int (2 ^Int 160)
100       andBool 0 <=Int CALLER_ID  andBool CALLER_ID  <Int (2 ^Int 160)
101       andBool 0 <=Int ORIGIN_ID  andBool ORIGIN_ID  <Int (2 ^Int 160)
102       andBool 0 <=Int CALL_DEPTH andBool CALL_DEPTH <Int 1024
103       andBool 0 <=Int OWNER      andBool OWNER      <Int (2 ^Int 160)
104   andBool 0 <=Int BAL        andBool BAL        <Int (2 ^Int 256)
105
```

ERC20 claim in KEVM

runtime verification

```
81          <accounts>
82            <account>
83              <acctID> ACCT_ID </acctID>
84              <balance> _ </balance>
85              <code> #parseByteStack("0x6060604052600436106100d576000357c0100000000000000000
86              <storage> #hashedLocation("Solidity", 1, OWNER) |-> BAL
87  _:Map </storage>
88              <origStorage> _ </origStorage>
89              <nonce> _ </nonce>
90            </account>

96        </network>
97      </ethereum>
98      </kevm>
99      requires 0 <=Int ACCT_ID    andBool ACCT_ID    <Int (2 ^Int 160)
100       andBool 0 <=Int CALLER_ID  andBool CALLER_ID  <Int (2 ^Int 160)
101       andBool 0 <=Int ORIGIN_ID  andBool ORIGIN_ID  <Int (2 ^Int 160)
102       andBool 0 <=Int CALL_DEPTH andBool CALL_DEPTH <Int 1024
103       andBool 0 <=Int OWNER      andBool OWNER      <Int (2 ^Int 160)
104   andBool 0 <=Int BAL        andBool BAL        <Int (2 ^Int 256)
105
```

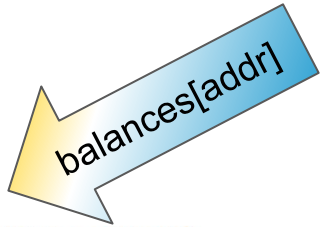
ERC20 claim in Solidity

runtime verification

```
67      function testBalanceOf(address addr, uint256 amount) public {
68          kevm.infiniteGas();
69          ERC20 erc20 = new ERC20("Bucharest Workshop Token", "BWT");
70          kevm.symbolicStorage(address(erc20));
71          bytes32 storageLocation = getStorageLocationForKey(addr, BALANCES_STORAGE_INDEX);
72          vm.assume(uint256(vm.load(address(erc20), storageLocation)) == amount);
73          uint256 balance = erc20.balanceOf(addr);
74          assertEq(balance, amount);
75      }
76
77      function getStorageLocationForKey(address _key, uint8 _index) public pure returns(bytes32) {
78          // Returns the index hash of the storage slot of a map at location `index` and the key `_
79          // returns `keccak(#buf(32,_key) +Bytes #buf(32, index))
80          return keccak256(abi.encode(_key, _index));
81      }
```

ERC20 claim in Solidity

runtime verification




```
67      function testBalanceOf(address addr, uint256 amount) public {
68          kevm.infiniteGas();
69          ERC20 erc20 = new ERC20("Bucharest Workshop Token", "BWT");
70          kevm.symbolicStorage(address(erc20));
71          bytes32 storageLocation = getStorageLocationForKey(addr, BALANCES_STORAGE_INDEX);
72          vm.assume(uint256(vm.load(address(erc20), storageLocation)) == amount);
73          uint256 balance = erc20.balanceOf(addr);
74          assertEq(balance, amount);
75      }
76
77      function getStorageLocationForKey(address _key, uint8 _index) public pure returns(bytes32) {
78          // Returns the index hash of the storage slot of a map at location `index` and the key `_
79          // returns `keccak(#buf(32,_key) +Bytes #buf(32, index))
80          return keccak256(abi.encode(_key, _index));
81      }
```

ERC20 claim in Solidity

runtime verification


```
anvacaru@desktop foundry-demo$ kup update kevm
 ✅ Successfully updated 'kevm'.
anvacaru@desktop foundry-demo$ forge build
[⠒] Compiling...
[⠢] Compiling 13 files with 0.8.17
[⠃] Solc 0.8.17 finished in 1.12s
Compiler run successful
anvacaru@desktop foundry-demo$ kevm foundry-kompile out
WARNING 2023-03-23 10:00:39,719 kevm_pyk.__main__ - Ignoring command-line option: -O0
WARNING 2023-03-23 10:00:39,719 kevm_pyk.__main__ - Ignoring command-line option: -O1
WARNING 2023-03-23 10:00:39,720 kevm_pyk.__main__ - Ignoring command-line option: -O2
WARNING 2023-03-23 10:00:39,720 kevm_pyk.__main__ - Ignoring command-line option: -O3
/nix/store/xzyq7h4cmjkr596010w5x6icrc918cgr-python3-3.10.9/lib/python3.10/tempfile.py:860: ResourceWarning: Im
plicitly cleaning up <TemporaryDirectory '/tmp/tmp7108j_at'>
  _warnings.warn(warn_message, ResourceWarning)
/nix/store/xzyq7h4cmjkr596010w5x6icrc918cgr-python3-3.10.9/lib/python3.10/tempfile.py:860: ResourceWarning: Im
plicitly cleaning up <TemporaryDirectory '/tmp/tmpji_lc_gl'>
  _warnings.warn(warn_message, ResourceWarning)
/nix/store/xzyq7h4cmjkr596010w5x6icrc918cgr-python3-3.10.9/lib/python3.10/tempfile.py:860: ResourceWarning: Im
plicitly cleaning up <TemporaryDirectory '/tmp/tmpk9_n2tw5'>
  _warnings.warn(warn_message, ResourceWarning)
anvacaru@desktop foundry-demo$ kevm foundry-prove out --test ERC20Test.testBalanceOf
WARNING 2023-03-23 10:03:57,091 kevm_pyk.__main__ - Ignoring command-line option: --definition: /nix/store/rnb
mb430vf7rjss2m0a6q73wh6pmc01a-kevm-61036c1817771b207dc7cc45d449ab2b167c75d6/lib/kevm/foundry
WARNING 2023-03-23 10:08:07,063 pyk.kcfg.explore - Falling back to manual branch extraction ERC20Test.testBala
nceOf: b4d201..2cee23
PROOF PASSED: ERC20Test.testBalanceOf
anvacaru@desktop foundry-demo$
```

KEVM in your workflow

runtime verification

1. Write your smart contracts in Solidity
2. Write your tests in Solidity
3. Run tests for random inputs
4. Run the same tests for symbolic inputs
5. Tests pass. Hurray!
   Tests fail. Contact Runtime Verification

Benefits of using KEVM

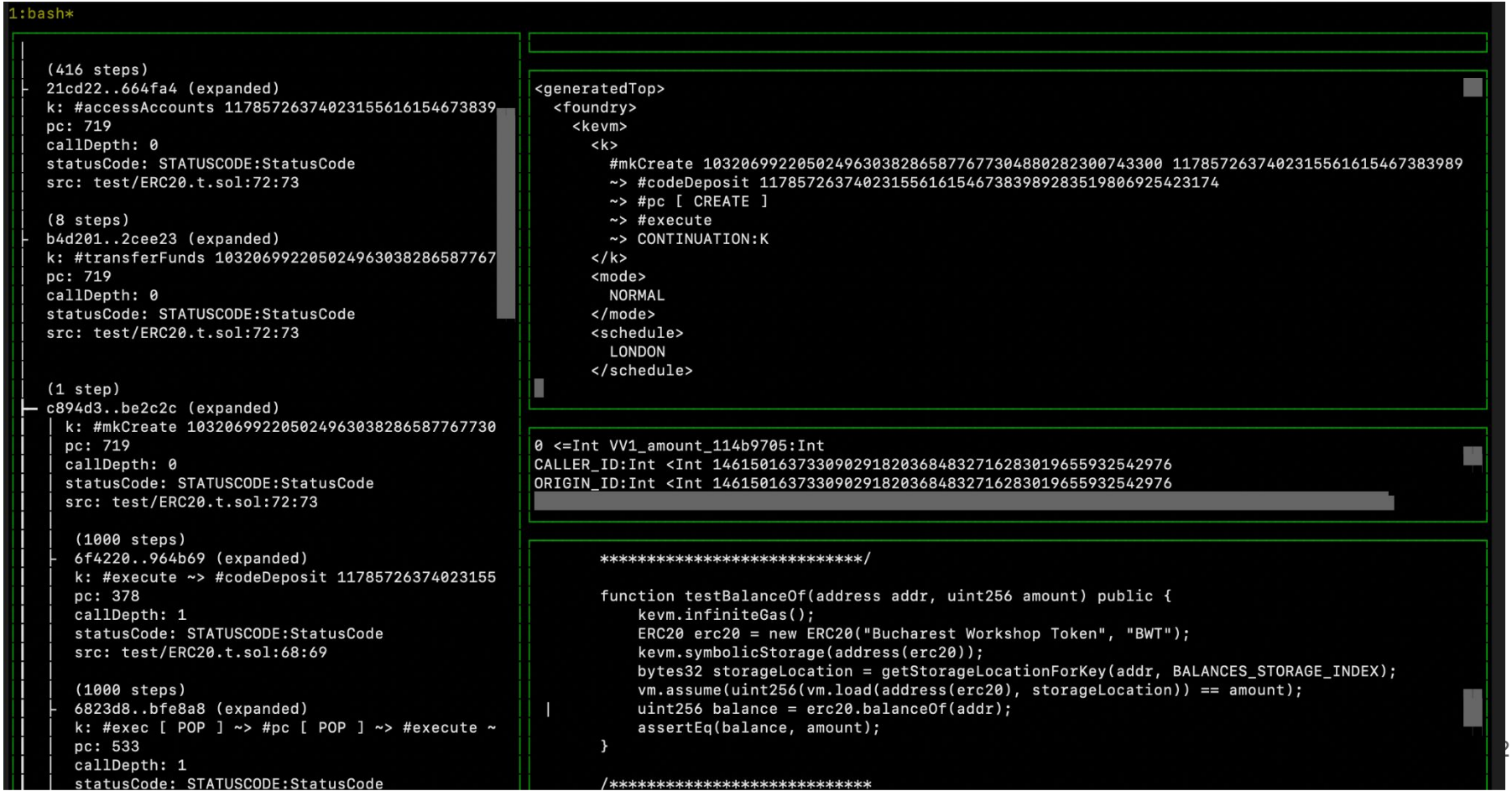
runtime verification

- Symbolic execution, which can be used to do formal verification of your property tests instead of fuzz testing (Much higher assurance that code is correct).

- Interactive debugger for stepping through program execution and visualizing the generated control flow graph.

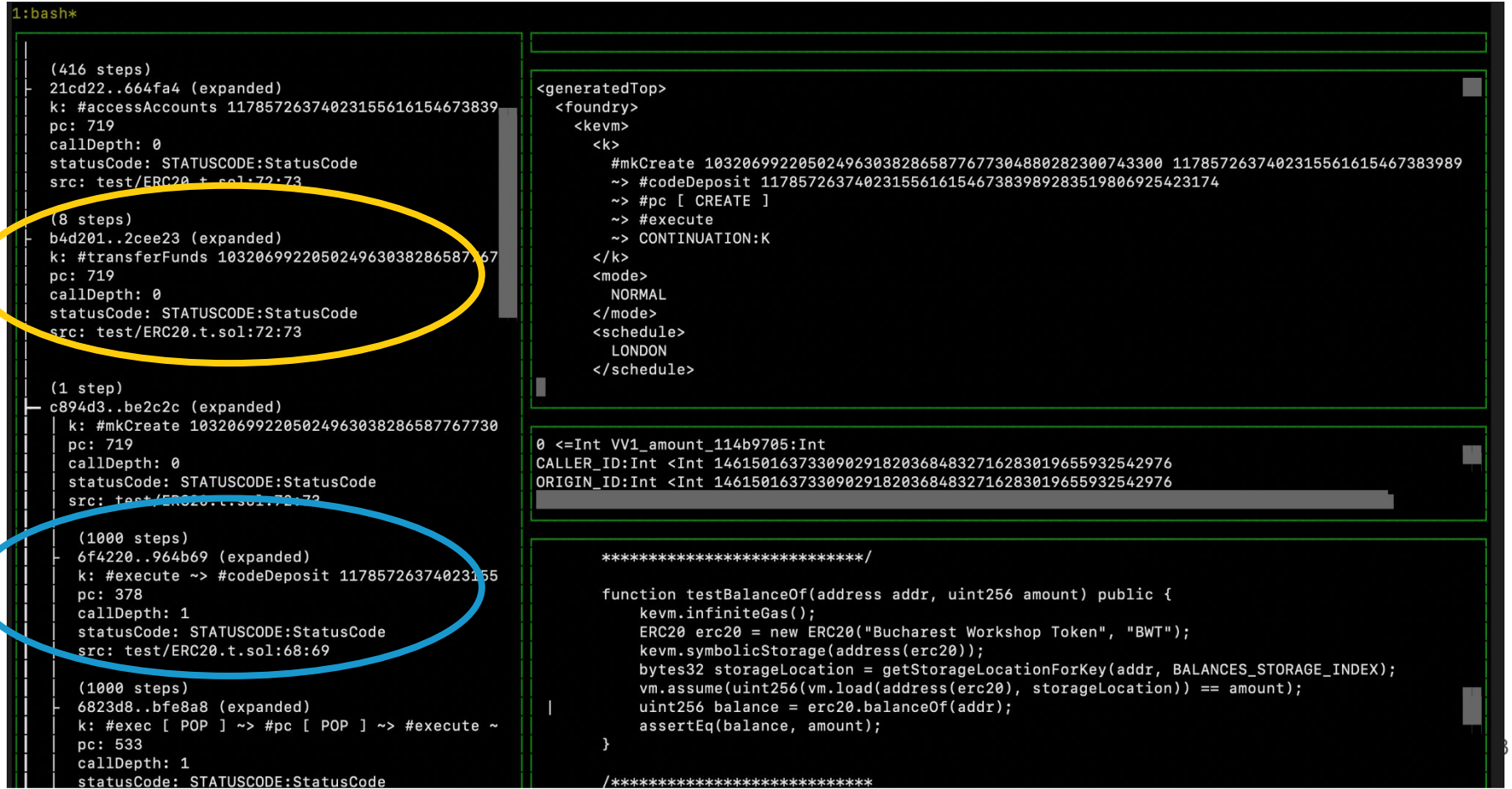
KEVM Interactive Debugger

runtime verification



KEVM Interactive Debugger

runtime verification



KEVM Interactive Debugger

runtime verification

- Simplifying nodes
- Removing nodes
- Resuming proofs
- Solidity source map integration

What's next?

runtime verification

- Improving the output and accessibility
- Support more tool kits
- Integration with other semantics
- Integration with tools such as ERCX

What's next?

runtime verification

- Hatom audit

  https://github.com/runtimeverification/publications/blob/main/reports/smart-contracts/Hatom-audit-report.pdf

- ESDT audit

  https://github.com/runtimeverification/publications/blob/main/reports/consensus-protocols/Elrond-ESDT.pdf

- Proofs for the MultiSig Wallet

  https://github.com/runtimeverification/elrond-multisig

Resources

runtime verification

Get started with KEVM Integration

Join our discord community

Thank you

runtime verification

Questions?