Invariant Testing WETH With Foundry

horsefacts.eth
@eth_call
terminally.online

EthCC[6]

Invariant Testing WETH With Foundry

horsefacts.eth
@eth_call
terminally.online

EthCC[6]

Setup

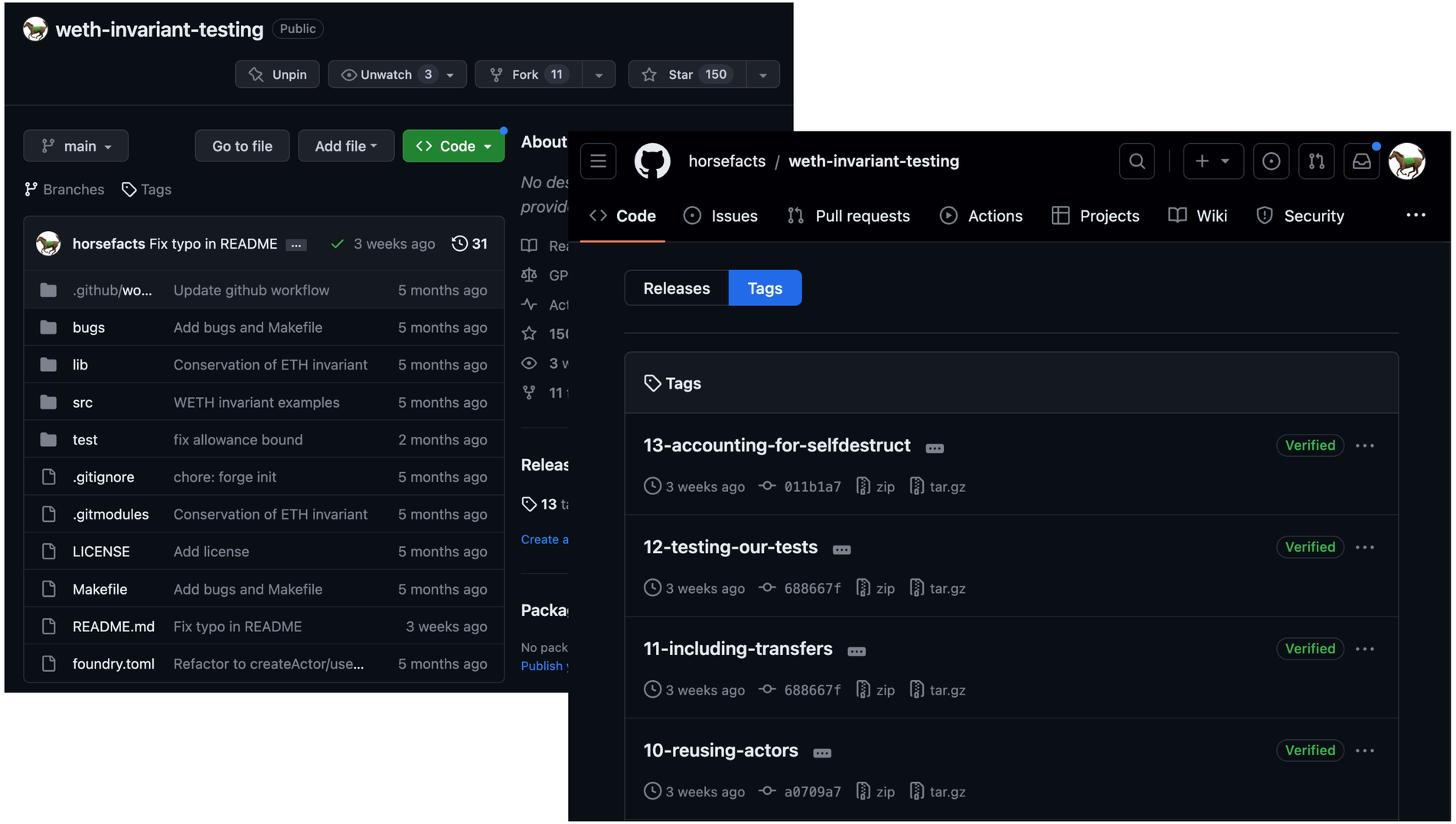
Install Foundry

```
$ curl -L https://foundry.paradigm.xyz | bash
$ foundryup
```

Clone Repo

```
$ git clone https://github.com/horsefacts/weth-invariant-testing.git
```

Setup


weth-invariant-testing
Public
Unpin
Unwatch 3
Fork 11
Star 150
main
Go to file
Add file
Code
Branches
Tags
horsefacts Fix typo in README
3 weeks ago
31
.github/wo... Update github workflow 5 months ago
bugs Add bugs and Makefile 5 months ago
lib Conservation of ETH invariant 5 months ago
src WETH invariant examples 5 months ago
test fix allowance bound 2 months ago
.gitignore chore: forge init 5 months ago
.gitmodules Conservation of ETH invariant 5 months ago
LICENSE Add license 5 months ago
Makefile Add bugs and Makefile 5 months ago
README.md Fix typo in README 3 weeks ago
foundry.toml Refactor to createActor/use... 5 months ago
About
horsefacts / weth-invariant-testing
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Releases
Tags
Tags
13-accounting-for-selfdestruct
Verified
3 weeks ago 011b1a7 zip tar.gz
12-testing-our-tests
Verified
3 weeks ago 688667f zip tar.gz
11-including-transfers
Verified
3 weeks ago 688667f zip tar.gz
10-reusing-actors
Verified
3 weeks ago a0709a7 zip tar.gz

Setup

Install Foundry

```
$ curl -L https://foundry.paradigm.xyz | bash
$ foundryup
```

Clone Repo

```
$ git clone https://github.com/horsefacts/weth-invariant-testing.git
```

How Invariant Tests Work

Unit tests: local behavior, specific input, expected output.

```
function test_addition() public {
    assertEq(5 + 3, 8);
}
```

How Invariant Tests Work

Fuzz tests: local behavior, random input, expected property.

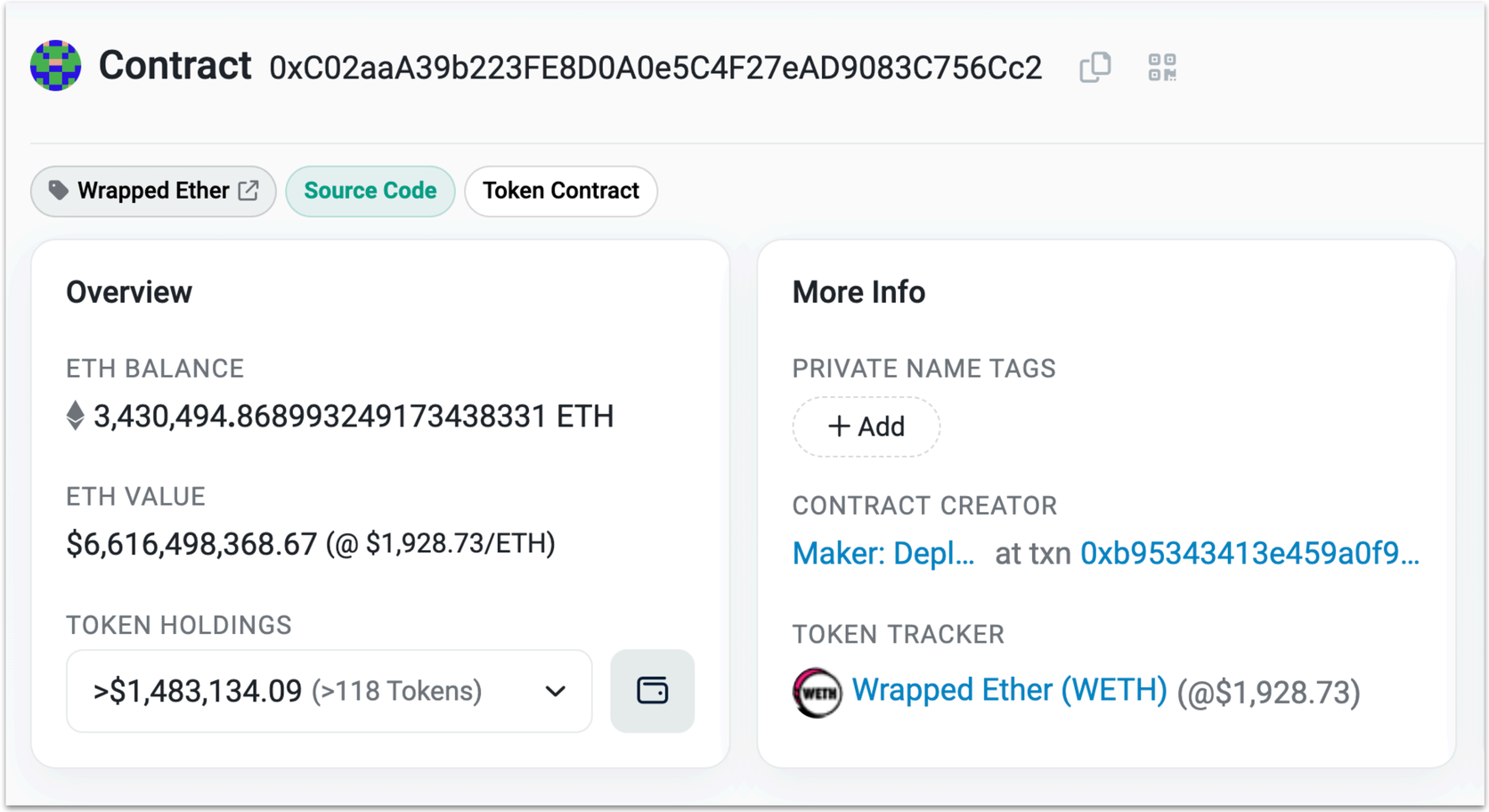
```
function test_addition_is_commutative(
    uint256 a,
    uint256 b
) public {
    assertEq(a + b, b + a);
}
```

How Invariant Tests Work

Invariant tests: system behavior, random input, expected property.

- "This vault contract always holds enough tokens to cover all withdrawals"
- "x * y always equals k in a Uniswap pool"
- “LPs can only withdraw the number of staking tokens they deposited”
- "This ERC20 token's supply always equals the sum of its individual balances"

Wrapped Ether



https://github.com/gnosis/canonical-weth/

Wrapped Ether

```
contract WETH9 {
    string public name = "Wrapped Ether";
    string public symbol = "WETH";
    uint8 public decimals = 18;

    event Approval(
        address indexed src,
        address indexed guy,
        uint256 wad
    );
    event Transfer(
        address indexed src,
        address indexed dst,
        uint256 wad
    );
    event Deposit(address indexed dst, uint256 wad);
    event Withdrawal(address indexed src, uint256 wad);
```

Metadata

Events

Wrapped Ether

Balances

```
mapping(address => uint256) public balanceOf;
mapping(address => mapping(address => uint256)) public allowance;

fallback() external payable {
    deposit();
}

function deposit() public payable {
    balanceOf[msg.sender] += msg.value;
    emit Deposit(msg.sender, msg.value);
}

function withdraw(uint256 wad) public {
    require(balanceOf[msg.sender] >= wad);
    balanceOf[msg.sender] -= wad;
    payable(msg.sender).transfer(wad);
    emit Withdrawal(msg.sender, wad);
}
```

Deposit/
Withdraw

Wrapped Ether

```
function totalSupply() public view returns (uint256) {
    return address(this).balance;
}

function approve(
    address guy,
    uint256 wad
) public returns (bool) {
    allowance[msg.sender][guy] = wad;
    emit Approval(msg.sender, guy, wad);
    return true;
}

function transfer(
    address dst,
    uint256 wad
) public returns (bool) {
    return transferFrom(msg.sender, dst, wad);
}
```

Transfers

Wrapped Ether

Transfers

```
    function transferFrom(
        address src,
        address dst,
        uint256 wad
    ) public returns (bool) {
        require(balanceOf[src] >= wad);
        if (
            src != msg.sender &&
            allowance[src][msg.sender] != type(uint256).max
        ) {
            require(allowance[src][msg.sender] >= wad);
            allowance[src][msg.sender] -= wad;
        }
        balanceOf[src] -= wad;
        balanceOf[dst] += wad;
        emit Transfer(src, dst, wad);
        return true;
    }
}
```

Test Setup

```
import {Test} from "forge-std/Test.sol";
import {WETH9} from "../src/WETH9.sol";

contract WETH9Invariants is Test {
    WETH9 public weth;

    function setUp() public {
        weth = new WETH9();
    }

    function invariant_wethSupplyIsZero() public {
        assertEq(weth.totalSupply(), 0);
    }
}
```

Test Setup

```
$ forge test
Running 1 test for test/
WETH9.invariants.t.sol:WETH9Invariants

[PASS] invariant_wethSupplyIsAlwaysZero()
(runs: 1000, calls: 15000, reverts: 8671)

Test result: ok. 1 passed; 0 failed; finished
in 873.42ms
```

Test Setup

```
$ forge test
Running 1 test for test/
WETH9.invariants.t.sol:WETH9Invariants

[PASS] invariant_wethSupplyIsAlwaysZero()
(runs: 1000, calls: 15000, reverts: 8671)

Test result: ok. 1 passed; 0 failed; finished
in 873.42ms
```

Test Setup

```
$ forge test
Running 1 test for test/
WETH9.invariants.t.sol:WETH9Invariants

[PASS] invariant_wethSupplyIsAlwaysZero()
(runs: 1000, calls: 15000, reverts: 8671)

Test result: ok. 1 passed; 0 failed; finished
in 873.42ms
```

Test Setup

```
$ forge test
Running 1 test for test/
WETH9.invariants.t.sol:WETH9Invariants

[PASS] invariant_wethSupplyIsAlwaysZero()
(runs: 1000, calls: 15000, reverts: 8671)

Test result: ok. 1 passed; 0 failed; finished
in 873.42ms
```

Test Setup

```
[invariant]
runs = 2000
depth = 25
fail_on_revert = false
call_override = false
dictionary_weight = 80
include_storage = true
include_push_bytes = true
```

Test Setup

```
[invariant]
runs = 2000
depth = 25
fail_on_revert = true
call_override = false
dictionary_weight = 80
include_storage = true
include_push_bytes = true
```

Test Setup

```
Failing tests:
[FAIL. Reason: EvmError: Revert]
  [Sequence]
    sender=0x0000000000000000000000000000000313ce567
    addr=[src/WETH9.sol:WETH9]
    calldata=approve(address,uint256):(bool),
    args=[0x0000000000000000000000000000000000000D65, 420]

    sender=0xda9208e3671a7222fb65771d8434a774ed6461fd
    addr=[src/WETH9.sol:WETH9]
    calldata=transferFrom(address,address,uint256):(bool),
    args=[0x53837153421b1bb193c40DebB360F09C5EFe9987,
          0x62Fb36f3b9487D04B85EFE8F5EbFEaC0DADb9ADD, 2]

invariant_wethSupplyIsAlwaysZero() (runs: 1, calls: 2, reverts: 1)
Encountered a total of 1 failing tests, 0 tests succeeded
```

Test Setup

```
Failing tests:
[FAIL. Reason: EvmError: Revert]
  [Sequence]
    sender=0x0000000000000000000000000000000313ce567
    addr=[src/WETH9.sol:WETH9]
    calldata=approve(address,uint256):(bool),
    args=[0x0000000000000000000000000000000000000D65, 420]

    sender=0xda9208e3671a7222fb65771d8434a774ed6461fd
    addr=[src/WETH9.sol:WETH9]
    calldata=transferFrom(address,address,uint256):(bool),
    args=[0x53837153421b1bb193c40DebB360F09C5EFe9987,
          0x62Fb36f3b9487D04B85EFE8F5EbFEaC0DADb9ADD, 2]

invariant_wethSupplyIsAlwaysZero() (runs: 1, calls: 2, reverts: 1)
Encountered a total of 1 failing tests, 0 tests succeeded
```

Handlers

```
contract Handler is CommonBase, StdCheats, StdUtils {
    WETH9 public weth;

    constructor(WETH9 _weth) {
        weth = _weth;
        deal(address(this), 10 ether);
    }

    function deposit(uint256 amount) public {
        weth.deposit{ value: amount }();
    }
}
```

Handlers

```
contract Handler is CommonBase, StdCheats, StdUtils {
    WETH9 public weth;

    constructor(WETH9 _weth) {
        weth = _weth;
        deal(address(this), 10 ether);
    }

    function deposit(uint256 amount) public {
        weth.deposit{ value: amount }();
    }
}
```

Handlers

```
contract WETH9Invariants is Test {
    WETH9 public weth;
    Handler public handler;

    function setUp() public {
        weth = new WETH9();
        handler = new Handler(weth);

        targetContract(address(handler));
    }

    function invariant_wethSupplyIsAlwaysZero() public {
        assertEq(0, weth.totalSupply());
    }
}
```

Handlers

```
$ forge test
Running 1 test for test/WETH9.invariants.t.sol:WETH9Invariants
Test result: FAILED. 0 passed; 1 failed; finished in 3.89ms

Failing tests:
Encountered 1 failing test in test/
WETH9.invariants.t.sol:WETH9Invariants
[FAIL. Reason: Assertion failed.]
        [Sequence]
                sender=0x00000000000000000000000000000000000000a4
                addr=[test/handlers/Handler.sol:Handler]
                     0x2e234dae75c793f67a35089c9d99245e1c58470b
                calldata=deposit(uint256),
                args=[65]

 invariant_wethSupplyIsAlwaysZero() (runs: 1, calls: 1, reverts: 0)
```

Handlers

```
$ forge test
Running 1 test for test/WETH9.invariants.t.sol:WETH9Invariants
Test result: FAILED. 0 passed; 1 failed; finished in 3.89ms

Failing tests:
Encountered 1 failing test in test/
WETH9.invariants.t.sol:WETH9Invariants
[FAIL. Reason: Assertion failed.]
        [Sequence]
                sender=0x00000000000000000000000000000000000000a4
                addr=[test/handlers/Handler.sol:Handler]
                     0x2e234dae75c793f67a35089c9d99245e1c58470b
                calldata=deposit(uint256),
                args=[65]

 invariant_wethSupplyIsAlwaysZero() (runs: 1, calls: 1, reverts: 0)
```

Handlers

```
$ forge test
Running 1 test for test/WETH9.invariants.t.sol:WETH9Invariants
Test result: FAILED. 0 passed; 1 failed; finished in 3.89ms

Failing tests:
Encountered 1 failing test in test/
WETH9.invariants.t.sol:WETH9Invariants
[FAIL. Reason: Assertion failed.]
        [Sequence]
                sender=0x00000000000000000000000000000000000000a4
                addr=[test/handlers/Handler.sol:Handler]
                     0x2e234dae75c793f67a35089c9d99245e1c58470b
                calldata=deposit(uint256),
                args=[65]

 invariant_wethSupplyIsAlwaysZero() (runs: 1, calls: 1, reverts: 0)
```

Handlers

```
$ forge test
Running 1 test for test/WETH9.invariants.t.sol:WETH9Invariants
Test result: FAILED. 0 passed; 1 failed; finished in 3.89ms

Failing tests:
Encountered 1 failing test in test/
WETH9.invariants.t.sol:WETH9Invariants
[FAIL. Reason: Assertion failed.]
        [Sequence]
                sender=0x00000000000000000000000000000000000000a4
                addr=[test/handlers/Handler.sol:Handler]
                     0x2e234dae75c793f67a35089c9d99245e1c58470b
                calldata=deposit(uint256),
                args=[65]

 invariant_wethSupplyIsAlwaysZero() (runs: 1, calls: 1, reverts: 0)
```

Handlers

```
$ forge test
Running 1 test for test/WETH9.invariants.t.sol:WETH9Invariants
Test result: FAILED. 0 passed; 1 failed; finished in 3.89ms

Failing tests:
Encountered 1 failing test in test/
WETH9.invariants.t.sol:WETH9Invariants
[FAIL. Reason: Assertion failed.]
        [Sequence]
                sender=0x00000000000000000000000000000000000000a4
                addr=[test/handlers/Handler.sol:Handler]
                     0x2e234dae75c793f67a35089c9d99245e1c58470b
                calldata=deposit(uint256),
                args=[65]

 invariant_wethSupplyIsAlwaysZero() (runs: 1, calls: 1, reverts: 0)
```

Conservation of Ether

```
// ETH can only be wrapped into WETH, WETH can only
// be unwrapped back into ETH. The sum of the Handler's
// ETH balance plus the WETH totalSupply() should always
// equal the total ETH_SUPPLY.
```

Conservation of Ether

```
contract Handler is CommonBase, StdCheats, StdUtils {
    WETH9 public weth;

    uint256 public constant ETH_SUPPLY = 120_500_000 ether;

    constructor(WETH9 _weth) {
        weth = _weth;
        deal(address(this), ETH_SUPPLY);
    }

    function deposit(uint256 amount) public {
        weth.deposit{ value: amount }();
    }
}
```

Conservation of Ether

```
contract Handler is CommonBase, StdCheats, StdUtils {
    WETH9 public weth;

    uint256 public constant ETH_SUPPLY = 120_500_000 ether;

    constructor(WETH9 _weth) {
        weth = _weth;
        deal(address(this), ETH_SUPPLY);
    }

    function deposit(uint256 amount) public {
        weth.deposit{ value: amount }();
    }
}
```

Conservation of Ether

```
// ETH can only be wrapped into WETH, WETH can only
// be unwrapped back into ETH. The sum of the Handler's
// ETH balance plus the WETH totalSupply() should always
// equal the total ETH_SUPPLY.
function invariant_conservationOfETH() public {
    assertEq(
      handler.ETH_SUPPLY(),
      address(handler).balance + weth.totalSupply()
    );
}
```

Conservation of Ether

```
contract Handler is CommonBase, StdCheats, StdUtils {
    WETH9 public weth;

    uint256 public constant ETH_SUPPLY = 120_500_000;

    constructor(WETH9 _weth) {
        weth = _weth;
        deal(address(this), ETH_SUPPLY);
    }

    function deposit(uint256 amount) public {
        amount = bound(amount, 0, address(this).balance);
        weth.deposit{ value: amount }();
    }
}
```

Conservation of Ether

```
$ forge test
Running 1 test for test/WETH9.invariants.t.sol:WETH9Invariants
[PASS] invariant_conservationOfETH()
(runs: 1000, calls: 15000, reverts: 0)
Test result: ok. 1 passed; 0 failed; finished in 1.24s
```

Adding Handler Functions

```
function withdraw(uint256 amount) public {
    amount = bound(amount, 0, weth.balanceOf(address(this)));
    weth.withdraw(amount);
}

receive() external payable {}
```

```
function sendFallback(uint256 amount) public {
    amount = bound(amount, 0, address(this).balance);
    (bool success,) = address(weth).call{ value: amount }("");
    require(success, "sendFallback failed");
}
```

Adding Handler Functions

```
$ forge test
Running 1 test for test/WETH9.invariants.t.sol:WETH9Invariants
[PASS] invariant_conservationOfETH()
(runs: 1000, calls: 15000, reverts: 0)
Test result: ok. 1 passed; 0 failed; finished in 1.24s
```

Solvency and Ghost Variables

```
// The WETH contract's Ether balance should always
// equal the sum of all the individual deposits
// minus all the individual withdrawals
```

Solvency and Ghost Variables

```
uint256 public ghost_depositSum;
uint256 public ghost_withdrawSum;

function deposit(uint256 amount) public {
    amount = bound(amount, 0, address(this).balance);
    weth.deposit{ value: amount }();
    ghost_depositSum += amount;
}

function withdraw(uint256 amount) public {
    amount = bound(amount, 0, weth.balanceOf(address(this)));
    weth.withdraw(amount);
    ghost_withdrawSum += amount;
}
```

Solvency and Ghost Variables

```
// The WETH contract's Ether balance should always
// equal the sum of all the individual deposits
// minus all the individual withdrawals
function invariant_solvencyDeposits() public {
    assertEq(
      address(weth).balance,
      handler.ghost_depositSum() - handler.ghost_withdrawSum()
    );
}
```

Solvency and Ghost Variables

```
$ forge test
[FAIL. Reason: Assertion failed.]
        [Sequence]
                sender=0x0000000000000000000000000000000000000c88
                calldata=deposit(uint256),
                  args=[826074471]
                sender=0x0000000000000000000000000000000000000ffb
                calldata=deposit(uint256),
                  args=[1]
                sender=0xeaae00d9e5544c3fd4fc519f81e2a4747920f369
                calldata=sendFallback(uint256),
                  args=[1007]

 invariant_solvencyDeposits()
 (runs: 1000, calls: 14988, reverts: 0)
Test result: FAILED. 1 passed; 1 failed; finished in 2.06s
```

Solvency and Ghost Variables

```
$ forge test
[FAIL. Reason: Assertion failed.]
        [Sequence]
                sender=0x0000000000000000000000000000000000000c88
                calldata=deposit(uint256),
                  args=[826074471]
                sender=0x0000000000000000000000000000000000000ffb
                calldata=deposit(uint256),
                  args=[1]
                sender=0xeaae00d9e5544c3fd4fc519f81e2a4747920f369
                calldata=sendFallback(uint256),
                  args=[1007]

 invariant_solvencyDeposits()
 (runs: 1000, calls: 14988, reverts: 0)
Test result: FAILED. 1 passed; 1 failed; finished in 2.06s
```

Solvency and Ghost Variables

```
function sendFallback(uint256 amount) public {
    amount = bound(amount, 0, address(this).balance);
    (bool success,) = address(weth).call{ value: amount }("");
    require(success, "sendFallback failed");
    ghost_depositSum += amount;
}
```

Solvency and Ghost Variables

```
$ forge test
Running 2 tests for test/
WETH9.invariants.t.sol:WETH9Invariants
[PASS] invariant_conservationOfETH()
(runs: 1000, calls: 15000, reverts: 0)
[PASS] invariant_solvencyDeposits()
(runs: 1000, calls: 15000, reverts: 0)
Test result: ok. 2 passed; 0 failed; finished in 2.18s
```

Solvency of Balances

```
// The WETH contract's Ether balance should always be
// at least as much as the sum of individual balances
function invariant_solvencyBalances() public {
    uint256 sumOfBalances = ???
    assertEq(
        address(weth).balance,
        sumOfBalances
    );
}
```

Introducing Actors

```
function deposit(uint256 amount) public {
    amount = bound(amount, 0, address(this).balance);
    _pay(msg.sender, amount);

    vm.prank(msg.sender);
    weth.deposit{value: amount}();

    ghost_depositSum += amount;
}

function _pay(address to, uint256 amount) internal {
    (bool s,) = to.call{value: amount}("");
    require(s, "pay() failed");
}
```

Introducing Actors

```
function withdraw(uint256 amount) public {
    amount = bound(amount, 0, weth.balanceOf(msg.sender));

    vm.startPrank(msg.sender);
    weth.withdraw(amount);
    _pay(address(this), amount);
    vm.stopPrank();

    ghost_withdrawSum += amount;
}
```

Introducing Actors

```
function sendFallback(uint256 amount) public {
    amount = bound(amount, 0, address(this).balance);
    _pay(msg.sender, amount);

    vm.prank(msg.sender);
    (bool success,) = address(weth).call{value: amount}("");

    require(success, "sendFallback failed");
    ghost_depositSum += amount;
}
```

Introducing Actors

```
$ forge test
Running 2 tests for test/
WETH9.invariants.t.sol:WETH9Invariants
[PASS] invariant_conservationOfETH()
(runs: 10000, calls: 150000, reverts: 0)
[PASS] invariant_solvencyDeposits()
(runs: 10000, calls: 150000, reverts: 0)
Test result: ok. 2 passed; 0 failed; finished in 95.41s
```

Using Actors

```
struct AddressSet {
    address[] addrs;
    mapping(address => bool) saved;
}
```

Using Actors

```
library LibAddressSet {
    function add(
        AddressSet storage s,
        address addr
    ) internal {
        if (!s.saved[addr]) {
            s.addrs.push(addr);
            s.saved[addr] = true;
        }
    }

    function count(
        AddressSet storage s
    ) internal view returns (uint256) {
        return s.addrs.length;
    }
```

Using Actors

```
    function rand(
        AddressSet storage s,
        uint256 seed
    ) internal view returns (address) {
        if (s.addrs.length > 0) {
            return s.addrs[seed % s.addrs.length];
        } else {
            return address(0);
        }
    }
}
```

Using Actors

```
contract Handler is CommonBase, StdCheats, StdUtils {
    using LibAddressSet for AddressSet;

    AddressSet internal _actors;

    // Other handler stuff omitted here

    function actors() external returns (address[] memory) {
      return _actors.addrs;
    }
}
```

Using Actors

```
address internal currentActor;

modifier createActor() {
    currentActor = msg.sender;
    _actors.add(msg.sender);
    _;
}

modifier useActor(uint256 seed) {
    currentActor = _actors.rand(seed);
    _;
}
```

Using Actors

```
// The WETH contract's Ether balance should always be
// at least as much as the sum of individual balances
function invariant_solvencyBalances() public {
    uint256 sumOfBalances;
    address[] memory actors = handler.actors();
    for (uint256 i; i < actors.length; ++i) {
        sumOfBalances += weth.balanceOf(actors[i]);
    }
    assertEq(
        address(weth).balance,
        sumOfBalances
    );
}
```

Using Actors

```
function deposit(uint256 amount) public createActor {
    amount = bound(amount, 0, address(this).balance);
    _pay(currentActor, amount);

    ghost_depositSum += amount;

    vm.prank(currentActor);
    weth.deposit{ value: amount }();
}
```

Using Actors

```
function sendFallback(uint256 amount) public createActor {
        amount = bound(amount, 0, address(this).balance);
        _pay(currentActor, amount);

        ghost_depositSum += amount;

        vm.prank(currentActor);
        (bool success,) = address(weth).call{ value: amount }("");
        require(success, "sendFallback failed");
}
```

Using Actors

```
function withdraw(
    uint256 amount,
    uint256 seed
) public useActor(seed) {
    amount = bound(amount, 0, weth.balanceOf(currentActor));

    ghost_withdrawSum += amount;

    vm.startPrank(currentActor);
    weth.withdraw(amount);
    _pay(address(this), amount);
    vm.stopPrank();
}
```

Using Actors

```
$ forge test
Running 3 tests for test/
WETH9.invariants.t.sol:WETH9Invariants
[PASS] invariant_conservationOfETH()
(runs: 10000, calls: 150000, reverts: 10)
[PASS] invariant_solvencyBalances()
(runs: 10000, calls: 150000, reverts: 10)
[PASS] invariant_solvencyDeposits()
(runs: 10000, calls: 150000, reverts: 10)
Test result: ok. 3 passed; 0 failed; finished in 134.45s
```

Adding Transfers

```
function approve(
    uint256 seed,
    uint256 spenderSeed,
    uint256 amount
) public useActor(seed) {
    address spender = _actors.rand(spenderSeed);

    vm.prank(currentActor);
    weth.approve(spender, amount);
}
```

Adding Transfers

```
function transfer(
    uint256 seed,
    uint256 toSeed,
    uint256 amount
) public useActor(seed) {
    address to = _actors.rand(toSeed);

    amount = bound(amount, 0, weth.balanceOf(currentActor));

    vm.prank(currentActor);
    weth.transfer(to, amount);
}
```

Adding Transfers

```
function transferFrom(
    uint256 seed,
    uint256 fromSeed,
    uint256 toSeed,
    bool _approve,
    uint256 amount
) public useActor(seed) {
    address from = _actors.rand(fromSeed);
    address to = _actors.rand(toSeed);
    amount = bound(amount, 0, weth.balanceOf(from));
    if (_approve) {
        vm.prank(from);
        weth.approve(currentActor, amount);
    } else {
        amount = bound(amount, 0, weth.allowance(currentActor, from));
    }
    vm.prank(currentActor);
    weth.transferFrom(from, to, amount);
}
```

Adding Transfers

```
$ forge test
Running 3 tests for test/
WETH9.invariants.t.sol:WETH9Invariants
[PASS] invariant_conservationOfETH()
(runs: 10000, calls: 150000, reverts: 10)
[PASS] invariant_solvencyBalances()
(runs: 10000, calls: 150000, reverts: 10)
[PASS] invariant_solvencyDeposits()
(runs: 10000, calls: 150000, reverts: 10)
Test result: ok. 3 passed; 0 failed; finished in 134.45s
```

Testing Our Tests

```
$ git diff > bugs/bug1.patch
$ cat bugs/bug1.patch
diff --git a/src/WETH9.sol b/src/WETH9.sol
index cd55b98..ccb40cb 100644
--- a/src/WETH9.sol
+++ b/src/WETH9.sol
@@ -33,7 +33,7 @@ contract WETH9 {
     }

     function deposit() public payable {
-        balanceOf[msg.sender] += msg.value;
+        balanceOf[msg.sender] += 1;
         emit Deposit(msg.sender, msg.value);
     }
```

Testing Our Tests

```
diff --git a/src/WETH9.sol b/src/WETH9.sol
index cd55b98..961f03b 100644
--- a/src/WETH9.sol
+++ b/src/WETH9.sol
@@ -40,7 +40,7 @@ contract WETH9 {
     function withdraw(uint256 wad) public {
         require(balanceOf[msg.sender] >= wad);
         balanceOf[msg.sender] -= wad;
-        payable(msg.sender).transfer(wad);
+        payable(msg.sender).transfer(1);
         emit Withdrawal(msg.sender, wad);
     }
```

Testing Our Tests

```
diff --git a/src/WETH9.sol b/src/WETH9.sol
index cd55b98..6e74bd5 100644
--- a/src/WETH9.sol
+++ b/src/WETH9.sol
@@ -29,7 +29,6 @@ contract WETH9 {
     mapping(address =>
         mapping(address => uint256)) public allowance;

     fallback() external payable {
-        deposit();
     }

     function deposit() public payable {
```

Testing Our Tests

```
check:
 git apply "bugs/$(bug).patch" && forge test

clean:
 git checkout src/WETH9.sol
```

```
$ make bug=bug1 check
$ make clean
```

Accounting for `selfdestruct`

```
function totalSupply() public view returns (uint256) {
    return address(this).balance;
}
```

Accounting for `selfdestruct`

```
contract ForcePush {
    constructor(address dst) payable {
        selfdestruct(payable(dst));
    }
}
```

```
function forcePush(
    uint256 amount
) public {
    amount = bound(amount, 0, address(this).balance);
    new ForcePush{ value: amount }(address(weth));
}
```

Accounting for `selfdestruct`

```
Running 3 tests for test/WETH9.invariants.t.sol:WETH9Invariants
[PASS] invariant_conservationOfETH()
[PASS] invariant_depositorBalances()
[FAIL. Reason: Assertion failed.]
        [Sequence]
                sender=0x0000000000000000000000000000000000000b69
                calldata=forcePush(uint256),
                args=[2250]
 invariant_solvencyBalances() (runs: 5000, calls: 74986, reverts: 9)
[FAIL. Reason: Assertion failed.]
        [Sequence]
                sender=0x0000000000000000000000000000000000000b69
                calldata=forcePush(uint256),
                args=[2250]
 invariant_solvencyDeposits() (runs: 5000, calls: 74986, reverts: 9)
Test result: FAILED. 2 passed; 2 failed; finished in 68.53s
```

Accounting for `selfdestruct`

```
Running 3 tests for test/WETH9.invariants.t.sol:WETH9Invariants
[PASS] invariant_conservationOfETH()
[PASS] invariant_depositorBalances()
[FAIL. Reason: Assertion failed.]
        [Sequence]
                sender=0x0000000000000000000000000000000000000b69
                calldata=forcePush(uint256),
                args=[2250]
 invariant_solvencyBalances() (runs: 5000, calls: 74986, reverts: 9)
[FAIL. Reason: Assertion failed.]
        [Sequence]
                sender=0x0000000000000000000000000000000000000b69
                calldata=forcePush(uint256),
                args=[2250]
 invariant_solvencyDeposits() (runs: 5000, calls: 74986, reverts: 9)
Test result: FAILED. 2 passed; 2 failed; finished in 68.53s
```

Accounting for `selfdestruct`

```
uint256 public ghost_forcePushSum;

function forcePush(
    uint256 amount
) public {
    amount = bound(amount, 0, address(this).balance);
    new ForcePush{ value: amount }(address(weth));
    ghost_forcePushSum += amount;
}
```

Accounting for `selfdestruct`

```
// The WETH contract's Ether balance should always be
// equal to the sum of all individual deposits
// minus all individual withrawals, plus any
// force-pushed Ether in the contract
function invariant_solvencyDeposits() public {
    assertEq(
        address(weth).balance,
        handler.ghost_depositSum() +
        handler.ghost_forcePushSum() -
        handler.ghost_withdrawSum()
    );
}
```

Accounting for `selfdestruct`

```
// The WETH contract's Ether balance should always be
// equal to the sum of individual balances plus any
// force-pushed Ether in the contract
function invariant_solvencyBalances() public {
    uint256 sumOfBalances = handler.reduceActors(
        0,
        this.accumulateBalance
    );
    assertEq(
        address(weth).balance - handler.ghost_forcePushSum(),
        sumOfBalances
    );
}
```

Accounting for `selfdestruct`

```
Running 3 tests for test/WETH9.invariants.t.sol:WETH9Invariants
[PASS] invariant_conservationOfETH()
(runs: 25000, calls: 625000, reverts: 15)
[PASS] invariant_solvencyBalances()
(runs: 25000, calls: 625000, reverts: 15)
[PASS] invariant_solvencyDeposits()
(runs: 25000, calls: 625000, reverts: 15)
Test result: ok. 4 passed; 0 failed; finished in 6995.00s
```

Merci beaucoup!

https://github.com/horsefacts/weth-invariant-testing

@eth_call