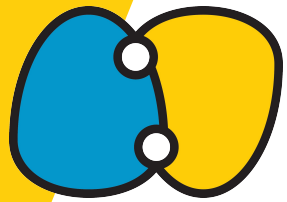
runtime verification



Kontrol UnlocKed

Foundry-based Formal Verification for 10x Devs and Auditors

Juan Conejero

Runtime Verification Inc.

About Us

runtime verification

Runtime Verification

Juan Conejero

Getting Kontrol

runtime verification

- Via `kup` (most user friendly):
  - `bash <(curl https://kframework.org/install)`
  - `kup install kontrol`
  - `kup` repo: https://github.com/runtimeverification/kup

Reproduction Steps

runtime verification

Workshop repository:

github.com/runtimeverification/kontrol-dss-2024

runtime verification

Your Kontrol Journey Starts Here.

💪⚒️💪 Ripped Jesus 💪⚒️💪

runtime verification

Foundry 🤝 Kontrol

runtime verification

```
function testMulWad(uint256 x, uint256 y) public {
    // No overflow case
    if (y == 0 || x <= type(uint256).max / y) {
        uint256 zSpec = (x * y) / WAD;
        uint256 zImpl = mulWad(x, y);
        // mulWad behaves as specified
        assert(zImpl == zSpec);
    } else {
        // If overflow, it should revert
        vm.expectRevert(MulWadFailed.selector);
        // External call needed for expectRevert to kick in
        this.mulWad(x, y);
    }
}
```

- You can run it as a Kontrol proof ⚙️
- Executing it symbolically instead of concretely 🧮

Run Kontrol

runtime verification

Swap “forge” with “kontrol” and “test” with “prove”:

```
forge build
```

↓

```
kontrol build
```

```
forge test
```

↓

```
kontrol prove
```

🏃 Run Kontrol 🏃

runtime verification

Got `testFoo`? You can `proveFoo` and `checkFoo` too!

Three proof prefixes:

- `test` (foundry)
- `prove` (hevm)
- `check` (halmos)

🍭 Cheatcodes 🍭

runtime verification

- Kontrol is kompatible with Foundry cheatcodes
- Tests enhanced by Kontrol’s own cheatcodes
- Foundry included some of Kontrol’s cheatcodes recently!

🍭 Cheatcodes 🍭

runtime verification

- Compatible with Foundry cheatcodes
  - Most execution-related cheatcodes work in Kontrol
  - Reach out to us if you’re missing a cheatcode!
  - Check out which cheatcodes have been implemented here

🍭 Cheatcodes 🍭

runtime verification

- Tests enhanced by Kontrol's own cheatcodes
  - We have our own symbolically-oriented cheatcodes
  - E.g.: `symbolicStorage(address)`, `fresh$Type`, `expect*Call`
  - Caution, these are not**(*)** executable with Foundry!

🍭 Cheatcodes 🍭

runtime verification

- Foundry included/adapted some of Kontrol's cheatcodes recently!
  - `setArbitraryStorage`
  - `copyStorage`
  - `mockFunction`
  - `random*`, where `*` ∈ {`Uint`, `Bool`, `Address`, `Bytes`}

🍭 Cheatcodes 🍭

runtime verification

- Kontrol Equivalents
  - `setArbitraryStorage` → `symbolicStorage`
  - `copyStorage` → Same
  - `mockFunction` → Same
  - `random*` → `fresh*`

🍭 Cheatcodes 🍭

runtime verification

- `setArbitraryStorage`
  - Foundry Version:

    Sets a **fixed (concrete) random** value to **unitialized** or **unread** storage slots

  - Kontrol Version (also called `symbolicStorage`):

    Sets the **entire storage** of an account as **symbolic**

🍭 Cheatcodes 🍭

runtime verification

- `copyStorage`
  - Foundry Version == Kontrol Version
    - Copies the entire storage from one account to another
    - Kontrol bonus: includes **transient storage**
    - Foundry caveat: won't work if `setArbitraryStorage` has been used

🍭 Cheatcodes 🍭

runtime verification

- `mockFunction`
  - Foundry Version == Kontrol Version
  - Enchances the existing `mockCall` cheatcode:
    - `mockCall` swaps a call with a **specified return value**
    - `mockFunction` swaps a call with a **specified function call**

🍭 Cheatcodes 🍭

runtime verification

- `random*`, where `*` ∈ {`Uint`, `Bool`, `Address`, `Bytes`}
  - Foundry Version:

    Outputs a **new fixed (concrete) random** value of the desired type
  - Kontrol version: `fresh*`, where `*` ∈ {`Uint`, `Bool`, `Address`, `Bytes`}

    Outputs a **new symbolic** value of the desired type

🍭 Cheatcodes 🍭

runtime verification

- Not (yet) implemented in Foundry:
  - `expectRegularCall`
  - `expectStaticCall`
  - `expectDelegateCall`
  - `expectNoCall`
  - `expectCreate`
  - `expectCreate2`

🤡 Why so symbolic? 🤡

runtime verification

🤡 Why so symbolic? 🤡

runtime verification

🤡 Why so symbolic? 🤡

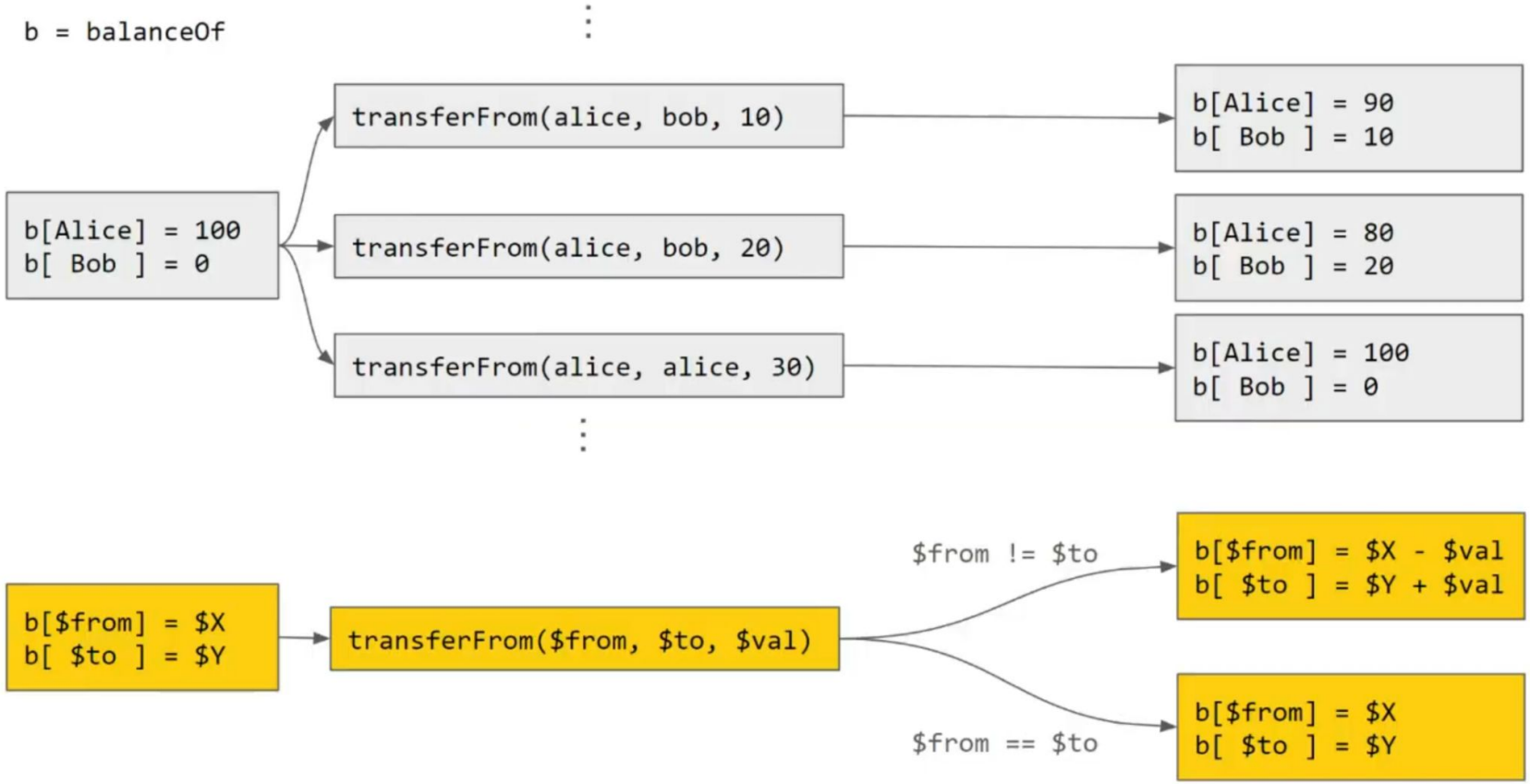
runtime verification

```
function setUp() public {
    token = new Token();
    vm.setArbitraryStorage(address(token));
}

// ∀ to, ∀ value, the transfer(to, value) will revert if the sender is address(0).
function test_transfer_RevertsWhenCallerIsZero(address to, uint256 value) public
{
    vm.startPrank(address(0));
    vm.expectRevert(abi.encodeWithSelector(IERC20Errors.ERC20InvalidSender.selector, address(0)));
    token.transfer(to, value);
}
```

What is Symbolic Execution?

runtime verification



What is Symbolic Execution?

runtime verification

- What is symbolic execution?
  - Execution
  - Symbolic
    - Not concrete

- Great. Let's elaborate:
  - `transferFrom(address from, address to, uint256 val)`
  - Concrete execution =
    - Execute with concrete `from`, `to` and `val`
    - Obtain concrete state update
  - Symbolic execution =
    - Execute with symbolic `from`, `to` or `val`
    - Obtain symbolic state update

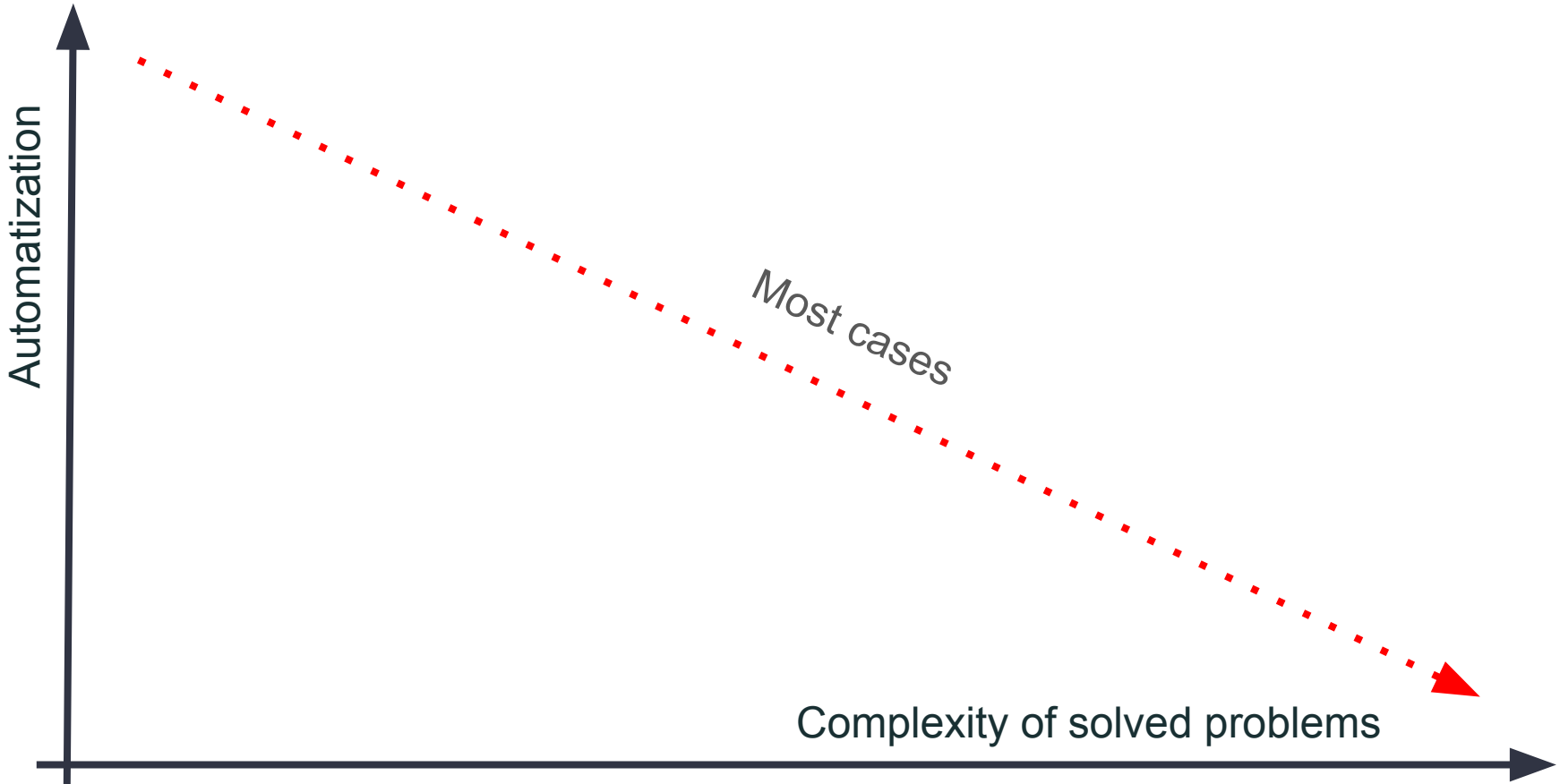
A Word on Formal Verification

runtime verification

- What is Formal Verification?
  - Assert properties of your code (i.e. **specify** its behavior)
  - Prove that your code satisfies the specification
- There are a lot of different techniques to achieve this
- Symbolic execution is one of them
  - Execute your code with symbolic parameters
  - Get all possible execution traces
  - Check that each execution trace satisfies the specification

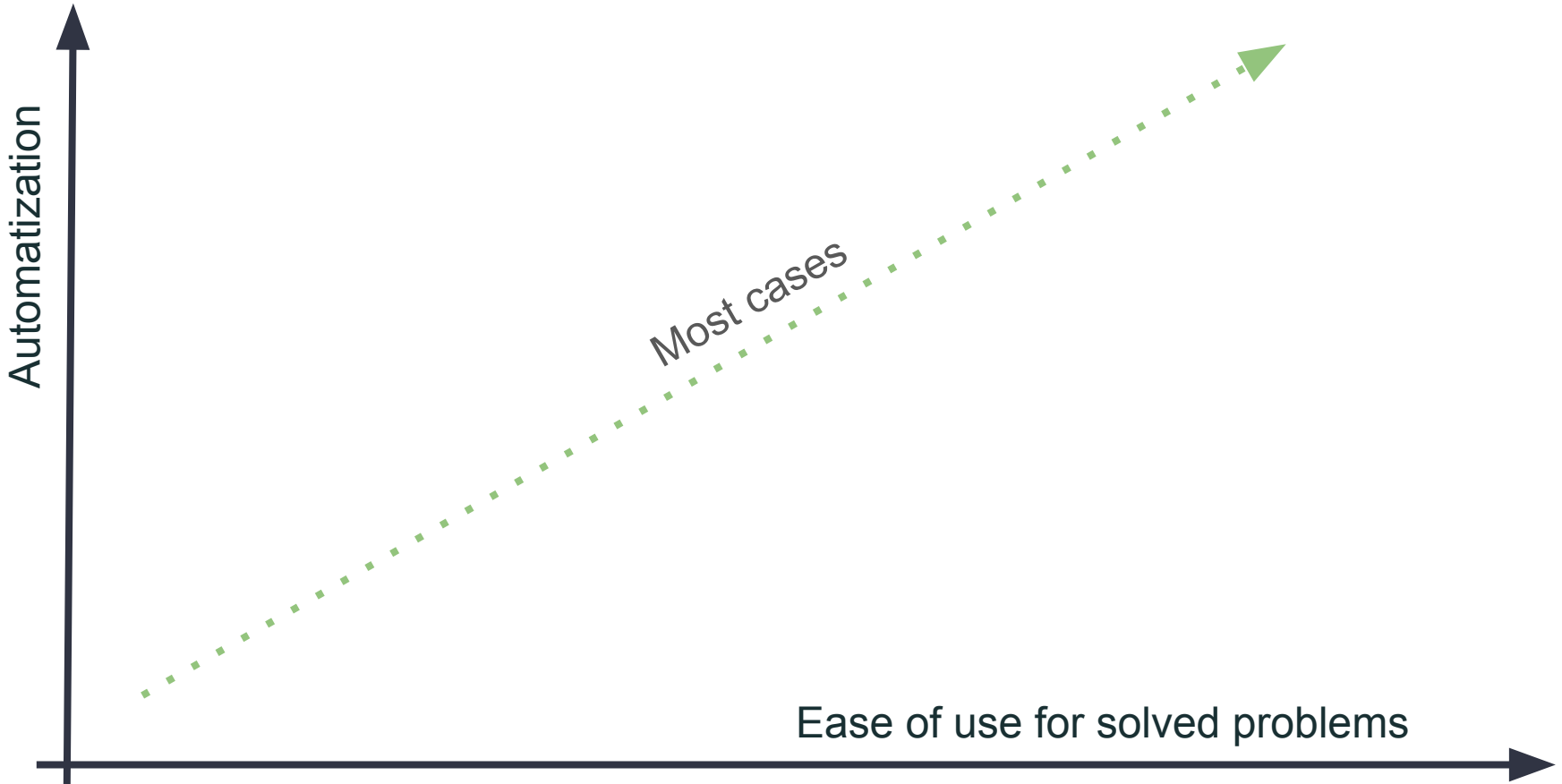
A Word on Formal Verification

runtime verification



A Word on Formal Verification

runtime verification



A Word on Formal Verification

runtime verification

How different proof systems look like:

- Interactive Theorem Proving (ITP)
  - Mostly manual
  - Constant feedback from the machine
- Automated Reasoning
  - No user interaction required
  - Usually no insights on the proving process

A Word on Formal Verification

runtime verification

Example of an ITP proof (LEAN4):

```
lemma IsHamiltonian.connected (hG : G.IsHamiltonian) : G.Connected where
  preconnected a b := by
    obtain rfl | hab := eq_or_ne a b
    · rfl
    have : Nontrivial α := ⟨a, b, hab⟩
    obtain ⟨_, p, hp⟩ := hG Fintype.one_lt_card.ne'
    have a_mem := hp.mem_support a
    have b_mem := hp.mem_support b
    exact ((p.takeUntil a a_mem).reverse.append <| p.takeUntil b b_mem).reachable
  nonempty := not_isEmpty_iff.1 fun _ ↦ by simpa using hG <| by simp [@Fintype.card_eq_zero]
```

A Word on Formal Verification

runtime verification

Examples of ITP proof feedback (LEAN4)

```
α : Type u_1
inst¹ : DecidableEq α
G : SimpleGraph α
inst : Fintype α
hG : G.IsHamiltonian
a b : α
hab : a ≠ b
this : Nontrivial α
w : α
p : G.Walk w w
hp : p.IsHamiltonianCycle
a_mem : a ∈ p.support
b_mem : b ∈ p.support
⊢ ∀ {V : Type u_1} {G : SimpleGraph V} {u v : V}, G.Walk u v → G.Reachable u v
```

```
case inr
α : Type u_1
inst¹ : DecidableEq α
G : SimpleGraph α
inst : Fintype α
hG : G.IsHamiltonian
a b : α
hab : a ≠ b
this : Nontrivial α
⊢ G.Reachable a b
```

A Word on Formal Verification

runtime verification

Example of an automated proof (halmos)

```
Running 1 tests for src/GaussSpec.sol:GaussSpec
[PASS] check_sumToN_success(uint256) (paths: 3, time: 0.02s, bounds: [])
WARNING:Halmos:check_sumToN_success(uint256): paths have not been fully explored due to the loop unrolling bound: 2
(see https://github.com/a16z/halmos/wiki/warnings#loop-bound)
Symbolic test result: 1 passed; 0 failed; time: 0.03s
```

Possible outcomes:

- Success
- Failure (with counterexample)
- Unknown (indeterminate)

A Word on Formal Verification

runtime verification

Example of a Kontrol proof: interactively automated

```
🏃 Running Kontrol proofs 🏃
 Add `--verbose` to `kontrol prove` for more details!
Selected functions: test%FixedPointMathLibVerification.testMulWad(uint256,uint256)
Running setup functions in parallel:

Running test functions in parallel: test%FixedPointMathLibVerification.testMulWad(uint256,uint256)
WARNING 2024-11-01 17:07:48,489 pyk.kcfg.kcfg - Extending current KCFG with the following: vacuous node: 20
test%FixedPointMathLibVerification.testMulWad(uint256… PENDING: 2 nodes: 1 pending|0 passed|0 failing|0 vacuous|0 refu…
✨ PROOF PASSED ✨ test%FixedPointMathLibVerification.testMulWad(uint256,uint256):0
⏳ Time: 1m 12s ⏳
```

A Word on Formal Verification

runtime verification

Feedback: inspecting symbolic execution traces in Kontrol

```
subst: .Subst
constraint:
    ( notBool VV1_y_114b9705:Int ==Int 0 )
    ( VV0_x_114b9705:Int *Int VV1_y_114b9705:Int ) <=Int maxUInt256
    ( notBool ( ( notBool VV0_x_114b9705:Int ==Int 0 ) andBool maxUInt256 /Word VV0_x_114b9705:Int <Int VV1_y_114b9705:Int ) )

35
 k: #execute ~> CONTINUATION:K
 pc: 0
 callDepth: 0
 statusCode: STATUSCODE:StatusCode
 src: test/FixedPointMathLib.k.sol:8:103
 method: test%FixedPointMathLibVerification.testMulWad(uint256,uint256)

(834 steps)
19 (terminal)
 k: #halt ~> CONTINUATION:K
 pc: 292
 callDepth: 0
 statusCode: EVMC_SUCCESS
 src: test/FixedPointMathLib.k.sol:73:102
 method: test%FixedPointMathLibVerification.testMulWad(uint256,uint256)

constraint: true
subst: ...
2 (leaf, target, terminal)
 k: #halt ~> CONTINUATION:K
 pc: PC_CELL_5d410f2a:Int
 callDepth: CALLDEPTH_CELL_5d410f2a:Int
 statusCode: STATUSCODE_FINAL:StatusCode
```

A Word on Formal Verification

runtime verification

Feedback: inspecting symbolic execution traces in Kontrol

```
subst: .Subst
constraint:
    ( notBool VV1_y_114b9705:Int ==Int 0 )
    ( VV0_x_114b9705:Int *Int VV1_y_114b9705:Int ) <=Int maxUInt256
    ( notBool ( ( notBool VV0_x_114b9705:Int ==Int 0 ) andBool maxUInt256 /Word VV0_x_114b9705:Int <Int VV1_y_114b9705:Int ) )
├─ 35
│   k: #execute ~> CONTINUATION:K
│   pc: 0
│   callDepth: 0
│   statusCode: STATUSCODE:StatusCode
│   src: test/FixedPointMathLib.k.sol:8:103
│   method: test%FixedPointMathLibVerification.testMulWad(uint256,uint256)
│
│  (834 steps)
├─ 19 (terminal)
│   k: #halt ~> CONTINUATION:K
│   pc: 292
│   callDepth: 0
│   statusCode: EVMC_SUCCESS
│   src: test/FixedPointMathLib.k.sol:73:102
│   method: test%FixedPointMathLibVerification.testMulWad(uint256,uint256)
│
┆  constraint: true
┆  subst: ...
└─ 2 (leaf, target, terminal)
    k: #halt ~> CONTINUATION:K
    pc: PC_CELL_5d410f2a:Int
    callDepth: CALLDEPTH_CELL_5d410f2a:Int
    statusCode: STATUSCODE_FINAL:StatusCode
```

A Word on Formal Verification

runtime verification

Feedback: inspecting symbolic execution traces in Kontrol

```
subst: .Subst
constraint:
    ( notBool VV1_y_114b9705:Int ==Int 0 )
    ( VV0_x_114b9705:Int *Int VV1_y_114b9705:Int ) <=Int maxUInt256
    ( notBool ( ( notBool VV0_x_114b9705:Int ==Int 0 ) andBool maxUInt256 /Word VV0_x_114b9705:Int <Int VV1_y_114b9705:Int ) )

35
 k: #execute ~> CONTINUATION:K
 pc: 0
 callDepth: 0
 statusCode: STATUSCODE:StatusCode
 src: test/FixedPointMathLib.k.sol:8:103
 method: test%FixedPointMathLibVerification.testMulWad(uint256,uint256)

(834 steps)
19 (terminal)
 k: #halt ~> CONTINUATION:K
 pc: 292
 callDepth: 0
 statusCode: EVMC_SUCCESS
 src: test/FixedPointMathLib.k.sol:73:102
 method: test%FixedPointMathLibVerification.testMulWad(uint256,uint256)

constraint: true
subst: ...
2 (leaf, target, terminal)
 k: #halt ~> CONTINUATION:K
 pc: PC_CELL_5d410f2a:Int
 callDepth: CALLDEPTH_CELL_5d410f2a:Int
 statusCode: STATUSCODE_FINAL:StatusCode
```

🤫 A Word on Formal Verification 🤫

runtime verification

Feedback: inspecting symbolic execution traces in Kontrol

```
<k>
  #halt
  ~> CONTINUATION:K
</k>
<mode>
  NORMAL
</mode>
<schedule>
  SHANGHAI
</schedule>
<useGas>
  false
</useGas>
<ethereum>
  <evm>
    <output>
      b""
    </output>
    <statusCode>
      EVMC_SUCCESS
    </statusCode>
    <callStack>
      .List
    </callStack>
```

```
<program>
  test%FixedPointMathLibVerification
</program>
<jumpDests>
  ...
</jumpDests>
<id>
  728815563385977040452943777879061427756277306518
</id>
<caller>
  CALLER_ID:Int
</caller>
<callData>
  b"\xd8?\xc1\r" +Bytes #buf ( 32 , VV0_x_114b9705:Int ) +Bytes #buf ( 32 , VV1_y_114b9705:Int )
</callData>
<callValue>
  0
</callValue>
<wordStack>
  ( selector ( "testMulWad(uint256,uint256)" ) : .WordStack )
</wordStack>
<localMem>
  b"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
\x00\x00\x00\x00\x00\x00\x00\x80"
</localMem>
<pc>
  292
</pc>
```

A Word on Formal Verification

runtime verification

Feedback: inspecting symbolic execution traces in Kontrol

```
<k>
  #halt
  ~> CONTINUATION:K
</k>
<mode>
  NORMAL
</mode>
<schedule>
  SHANGHAI
</schedule>
<useGas>
  false
</useGas>
<ethereum>
  <evm>
    <output>
      b""
    </output>
    <statusCode>
      EVMC_SUCCESS
    </statusCode>
    <callStack>
      .List
    </callStack>
```

```
<program>
  test%FixedPointMathLibVerification
</program>
<jumpDests>
  ...
</jumpDests>
<id>
  728815563385977040452943777879061427756277306518
</id>
<caller>
  CALLER_ID:Int
</caller>
<callData>
  b"\xd8?\xc1\r" +Bytes #buf ( 32 , VV0_x_114b9705:Int ) +Bytes #buf ( 32 , VV1_y_114b9705:Int )
</callData>
<callValue>
  0
</callValue>
<wordStack>
  ( selector ( "testMulWad(uint256,uint256)" ) : .WordStack )
</wordStack>
<localMem>
  b"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
\x00\x00\x00\x00\x00\x00\x00\x80"
</localMem>
<pc>
  292
</pc>
```

A Word on Formal Verification

runtime verification

Feedback: inspecting symbolic execution traces in Kontrol

```
<k>
  #halt
  ~> CONTINUATION:K
</k>
<mode>
  NORMAL
</mode>
<schedule>
  SHANGHAI
</schedule>
<useGas>
  false
</useGas>
<ethereum>
  <evm>
    <output>
      b""
    </output>
    <statusCode>
      EVMC_SUCCESS
    </statusCode>
    <callStack>
      .List
    </callStack>
```

```
<program>
  test%FixedPointMathLibVerification
</program>
<jumpDests>
  ...
</jumpDests>
<id>
  728815563385977040452943777879061427756277306518
</id>
<caller>
  CALLER_ID:Int
</caller>
<callData>
  b"\xd8?\xc1\r" +Bytes #buf ( 32 , VV0_x_114b9705:Int ) +Bytes #buf ( 32 , VV1_y_114b9705:Int )
</callData>
<callValue>
  0
</callValue>
<wordStack>
  ( selector ( "testMulWad(uint256,uint256)" ) : .WordStack )
</wordStack>
<localMem>
  b"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
\x00\x00\x00\x00\x00\x00\x00\x80"
</localMem>
<pc>
  292
</pc>
```

A Word on Formal Verification

runtime verification

Feedback: inspecting symbolic execution traces in Kontrol

```
<k>
  #halt
  ~> CONTINUATION:K
</k>
<mode>
  NORMAL
</mode>
<schedule>
  SHANGHAI
</schedule>
<useGas>
  false
</useGas>
<ethereum>
  <evm>
    <output>
      b""
    </output>
    <statusCode>
      EVMC_SUCCESS
    </statusCode>
    <callStack>
      .List
    </callStack>
```

```
<program>
  test%FixedPointMathLibVerification
</program>
<jumpDests>
  ...
</jumpDests>
<id>
  728815563385977040452943777879061427756277306518
</id>
<caller>
  CALLER_ID:Int
</caller>
<callData>
  b"\xd8?\xc1\r" +Bytes #buf ( 32 , VV0_x_114b9705:Int ) +Bytes #buf ( 32 , VV1_y_114b9705:Int )
</callData>
<callValue>
  0
</callValue>
<wordStack>
  ( selector ( "testMulWad(uint256,uint256)" ) : .WordStack )
</wordStack>
<localMem>
  b"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
\x00\x00\x00\x00\x00\x00\x00\x80"
</localMem>
<pc>
  292
</pc>
```

A Word on Formal Verification

runtime verification

Feedback: inspecting symbolic execution traces in Kontrol

```
<k>
  #halt
  ~> CONTINUATION:K
</k>
<mode>
  NORMAL
</mode>
<schedule>
  SHANGHAI
</schedule>
<useGas>
  false
</useGas>
<ethereum>
  <evm>
    <output>
      b""
    </output>
    <statusCode>
      EVMC_SUCCESS
    </statusCode>
    <callStack>
      .List
    </callStack>
```

```
<program>
  test%FixedPointMathLibVerification
</program>
<jumpDests>
  ...
</jumpDests>
<id>
  72881556338597704045294377787906142775627730651 8
</id>
<caller>
  CALLER_ID:Int
</caller>
<callData>
  b"\xd8?\xc1\r" +Bytes #buf ( 32 , VV0_x_114b9705:Int ) +Bytes #buf ( 32 , VV1_y_114b9705:Int )
</callData>
<callValue>
  0
</callValue>
<wordStack>
  ( selector ( "testMulWad(uint256,uint256)" ) : .WordStack )
</wordStack>
<localMem>
  b"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
\x00\x00\x00\x00\x00\x00\x00\x80"
</localMem>
<pc>
  292
</pc>
```

A Word on Formal Verification

runtime verification

Feedback: inspecting symbolic execution traces in Kontrol

```
<k>
  #halt
  ~> CONTINUATION:K
</k>
<mode>
  NORMAL
</mode>
<schedule>
  SHANGHAI
</schedule>
<useGas>
  false
</useGas>
<ethereum>
  <evm>
    <output>
      b""
    </output>
    <statusCode>
      EVMC_SUCCESS
    </statusCode>
    <callStack>
      .List
    </callStack>
```

```
<program>
  test%FixedPointMathLibVerification
</program>
<jumpDests>
  ...
</jumpDests>
<id>
  728815563385977040452943777879061427756277306518
</id>
<caller>
  CALLER_ID:Int
</caller>
<callData>
  b"\xd8?\xc1\r" +Bytes #buf ( 32 , VV0_x_114b9705:Int ) +Bytes #buf ( 32 , VV1_y_114b9705:Int )
</callData>
<callValue>
  0
</callValue>
<wordStack>
  ( selector ( "testMulWad(uint256,uint256)" ) : .WordStack )
</wordStack>
<localMem>
  b"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
\x00\x00\x00\x00\x00\x00\x00\x80"
</localMem>
<pc>
  292
</pc>
```

🤫 A Word on Formal Verification 🤫

Feedback: inspecting symbolic execution traces in Kontrol

```
<k>
  #halt
  ~> CONTINUATION:K
</k>
<mode>
  NORMAL
</mode>
<schedule>
  SHANGHAI
</schedule>
<useGas>
  false
</useGas>
<ethereum>
  <evm>
    <output>
      b""
    </output>
    <statusCode>
      EVMC_SUCCESS
    </statusCode>
    <callStack>
      .List
    </callStack>
```

```
<program>
  test%FixedPointMathLibVerification
</program>
<jumpDests>
  ...
</jumpDests>
<id>
  728815563385977040452943777879061427756277306518
</id>
<caller>
  CALLER_ID:Int
</caller>
<callData>
  b"\xd8?\xc1\r" +Bytes #buf ( 32 , VV0_x_114b9705:Int ) +Bytes #buf ( 32 , VV1_y_114b9705:Int )
</callData>
<callValue>
  0
</callValue>
<wordStack>
  ( selector ( "testMulWad(uint256,uint256)" ) : .WordStack )
</wordStack>
<localMem>
  b"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
\x00\x00\x00\x00\x00\x00\x00\x80"
</localMem>
<pc>
  292
</pc>
```

🤫 A Word on Formal Verification 🤫

runtime verification

Feedback: inspecting symbolic execution traces in Kontrol

```
<k>
  #halt
  ~> CONTINUATION:K
</k>
<mode>
  NORMAL
</mode>
<schedule>
  SHANGHAI
</schedule>
<useGas>
  false
</useGas>
<ethereum>
  <evm>
    <output>
      b""
    </output>
    <statusCode>
      EVMC_SUCCESS
    </statusCode>
    <callStack>
      .List
    </callStack>
```

```
<program>
  test%FixedPointMathLibVerification
</program>
<jumpDests>
  ...
</jumpDests>
<id>
  728815563385977040452943777879061427756277306518
</id>
<caller>
  CALLER_ID:Int
</caller>
<callData>
  b"\xd8?\xc1\r" +Bytes #buf ( 32 , VV0_x_114b9705:Int ) +Bytes #buf ( 32 , VV1_y_114b9705:Int )
</callData>
<callValue>
  0
</callValue>
<wordStack>
  ( selector ( "testMulWad(uint256,uint256)" ) : .WordStack )
</wordStack>
<localMem>
  b"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
\x00\x00\x00\x00\x00\x00\x00\x80"
</localMem>
<pc>
  292
</pc>
```

A Word on Formal Verification

runtime verification

Interactiveness: enhancing Kontrol's reasoning power

```
requires "foundry.md"

module MY-LEMMAS
    imports BOOL
    imports FOUNDRY
    imports INT-SYMBOLIC

    rule bool2Word ( X ) => 1 requires X         [simplification]
    rule bool2Word ( X ) => 0 requires notBool X [simplification]

    rule chop ( bool2Word ( X ) <<Int Y ) => bool2Word ( X ) <<Int Y requires Y <Int 256 [simplification]

    rule chop ( 1 <<Int Y ) => 1 <<Int Y requires Y <Int 256 [simplification]

    rule ( A:Int >>Int B:Int ) >>Int C:Int => A >>Int ( B +Int C ) [simplification]

    rule A =/=Int B => notBool ( A ==Int B ) [simplification, comm]

endmodule
```

Formal Verification in Ethereum

runtime verification

Non-exhaustive list of maintained Formal Verification EVM tools:

- **CLEAR** (Nethermind) [Type Theory]
- **coq-of-solidity** (Formal Land) [Type Theory]
- **hevm** (Ethereum Foundation) [SMT-Based]
- **Certora Prover** (Certora) [SMT-Based]
- **Halmos** (a16z) [SMT-Based]
- **Kontrol** (Runtime Verification) [Rewrite-Based]

Get excited for Kontrol!

runtime verification

All fine and dandy

but this is a workshop

after all…

runtime verification

Verifying Stuff with Kontrol.

Hardware Requirements

runtime verification

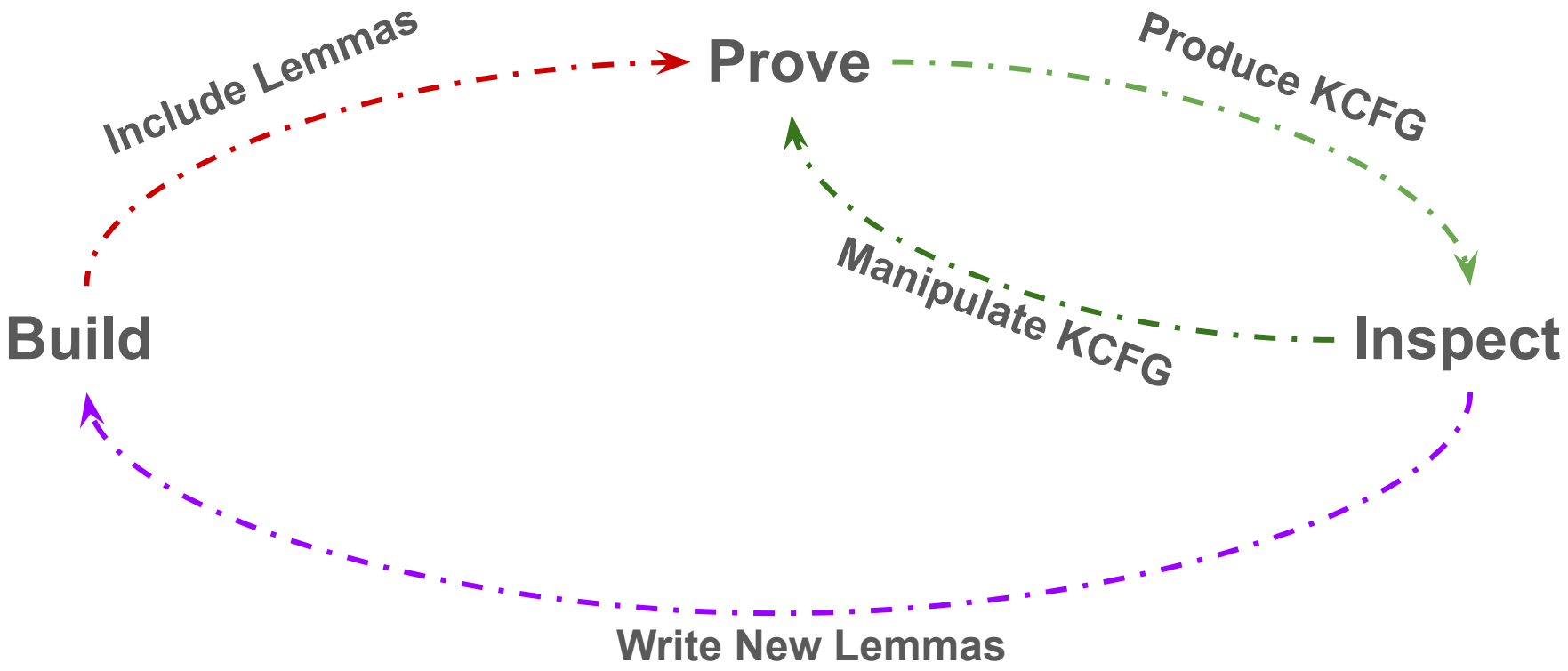
- Minimum 16GB RAM to build projects
  - Windows Subsystem for Linux (wsl): Default memory is ~2GB, increase
  - Docker container: Make sure your machine allocated enough memory
- Running proofs in parallel
  - As a rule of thumb, consider 8GB RAM per parallel proof
  - This rule can be relaxed: experiment

first_verification_attempt.jpeg

runtime verification

Kontrol Prove Interaction Loop

runtime verification



Kontrol à la carte

runtime verification

kontrol build options

```
kontrol build [-h] [--verbose] [--debug] [-I INCLUDES]
              [--main-module MAIN_MODULE]
              [--syntax-module SYNTAX_MODULE]
              [--md-selector MD_SELECTOR] [--depth DEPTH]
              [--require REQUIRES] [--module-import IMPORTS]
              [--output-definition DEFINITION_DIR] [--backend BACKEND]
              [--type-inference-mode TYPE_INFERENCE_MODE] [--emit-json]
              [-ccopt CCOPTS] [--no-llvm-kompile] [--with-llvm-library]
              [--enable-llvm-debug]
              [--llvm-kompile-type LLVM_KOMPILE_TYPE]
              [--llvm-kompile-output LLVM_KOMPILE_OUTPUT]
              [--read-only-kompiled-directory] [-O0] [-O1] [-O2] [-O3]
              [--enable-search] [--coverage] [--gen-bison-parser]
              [--gen-glr-bison-parser] [--bison-lists]
              [--llvm-proof-hint-instrumentation]
              [--llvm-proof-hint-debugging] [--no-exc-wrap]
              [--ignore-warnings IGNORE_WARNINGS]
              [--foundry-project-root FOUNDRY_ROOT]
              [--enum-constraints]
              [--target {KompileTarget.HASKELL,KompileTarget.MAUDE}]
              [--config-file CONFIG_FILE]
              [--config-profile CONFIG_PROFILE] [--regen] [--rekompile]
              [--no-forge-build] [--no-silence-warnings]
```

kontrol prove options

```
kontrol prove [-h] [--verbose] [--debug] [--workers WORKERS]
              [-I INCLUDES] [--main-module MAIN_MODULE]
              [--syntax-module SYNTAX_MODULE]
              [--md-selector MD_SELECTOR] [--depth DEPTH]
              [--debug-equations DEBUG_EQUATIONS]
              [--fast-check-subsumption] [--direct-subproof-rules]
              [--maintenance-rate MAINTENANCE_RATE] [--assume-defined]
              [--smt-timeout SMT_TIMEOUT]
              [--smt-retry-limit SMT_RETRY_LIMIT]
              [--smt-tactic SMT_TACTIC] [--no-log-rewrites]
              [--log-fail-rewrites]
              [--kore-rpc-command KORE_RPC_COMMAND] [--use-booster]
              [--no-use-booster] [--port PORT]
              [--maude-port MAUDE_PORT] [--bug-report BUG_REPORT]
              [--break-every-step] [--break-on-jumpi]
              [--break-on-calls] [--no-break-on-calls]
              [--break-on-storage] [--break-on-basic-blocks]
              [--symbolic-immutables] [--max-depth MAX_DEPTH]
              [--max-iterations MAX_ITERATIONS] [--failure-information]
              [--no-failure-information] [--auto-abstract-gas]
              [--counterexample-information]
              [--no-counterexample-information] [--fail-fast]
              [--force-sequential] [--no-fail-fast]
```

Kontrol build: the essentials

runtime verification

```
kontrol build [-h] [--verbose] [--debug] [-I INCLUDES]
              [--main-module MAIN_MODULE]
              [--syntax-module SYNTAX_MODULE]
              [--md-selector MD_SELECTOR] [--depth DEPTH]
              [--require REQUIRES] [--module-import IMPORTS]
              [--output-definition DEFINITION_DIR] [--backend BACKEND]
              [--type-inference-mode TYPE_INFERENCE_MODE] [--emit-json]
              [-ccopt CCOPTS] [--no-llvm-kompile] [--with-llvm-library]
              [--enable-llvm-debug]
              [--llvm-kompile-type LLVM_KOMPILE_TYPE]
              [--llvm-kompile-output LLVM_KOMPILE_OUTPUT]
              [--read-only-kompiled-directory] [-O0] [-O1] [-O2] [-O3]
              [--enable-search] [--coverage] [--gen-bison-parser]
              [--gen-glr-bison-parser] [--bison-lists]
              [--llvm-proof-hint-instrumentation]
              [--llvm-proof-hint-debugging] [--no-exc-wrap]
              [--ignore-warnings IGNORE_WARNINGS]
              [--foundry-project-root FOUNDRY_ROOT]
              [--enum-constraints]
              [--target {KompileTarget.HASKELL,KompileTarget.MAUDE}]
              [--config-file CONFIG_FILE]
              [--config-profile CONFIG_PROFILE] [--regen] [--rekompile]
              [--no-forge-build] [--no-silence-warnings]
```

Kontrol build: the essentials

runtime verification

Relevant options revolve around **lemmas** in the build process:

- Include **auxiliary** lemmas
- Include **custom** lemmas
- **Exclude keccak** lemmas
- Regenerate K definitions (used to include custom lemmas)

Kontrol build: the essentials

runtime verification

Include **auxiliary** lemmas

- `--auxiliary-lemmas`

```
rule { #buf( N, X:Int ) +Bytes B2:Bytes #Equals B } =>
       { X #Equals #asWord ( #range ( B, 0, N ) ) } #And
       { #range ( B, N, lengthBytes(B) -Int N ) #Equals B2 }
  requires N <=Int lengthBytes(B)
  [simplification(60), concrete(B, N)]
```

```
kontrol build --auxiliary-lemmas
```

Kontrol build: the essentials

runtime verification

Include **custom** lemmas

- `--require $path` Path to extra K files for the build process
- `--module-import $contract_name:$module_name`
  - Adds the specified module from the `required` K file
  - `$contract_name` Any contract that is compiled
  - `$module_name` K module to import from the required K file

```
kontrol build --auxiliary-lemmas --require $path --module-import
$contract_name:$module_name
```

Kontrol build: the essentials

runtime verification

Exclude keccak lemmas

- `--no-keccak-lemmas` Do not include assumptions on keccak properties
  - Injectivity (different objects have different keccaks)
  - Collision resistance for symbolic values
  - Some more

```
kontrol build --auxiliary-lemmas --require $path --module-import
$contract_name:$module_name
```

Kontrol build: the essentials

runtime verification

Regenerate K definitions (used to include more lemmas)

- `--regen` Regenerate `foundry.k` even if it already exists
- `--rekompile` Rekompile `foundry.k` even if kompiled definition exists

```
kontrol build --auxiliary-lemmas --require $path --module-import
$contract_name:$module_name --regen --rekompile
```

Kontrol prove: weaponry

runtime verification

```
kontrol prove [-h] [--verbose] [--debug] [--workers WORKERS]
              [-I INCLUDES] [--main-module MAIN_MODULE]
              [--syntax-module SYNTAX_MODULE]
              [--md-selector MD_SELECTOR] [--depth DEPTH]
              [--debug-equations DEBUG_EQUATIONS]
              [--fast-check-subsumption] [--direct-subproof-rules]
              [--maintenance-rate MAINTENANCE_RATE] [--assume-defined]
              [--smt-timeout SMT_TIMEOUT]
              [--smt-retry-limit SMT_RETRY_LIMIT]
              [--smt-tactic SMT_TACTIC] [--no-log-rewrites]
              [--log-fail-rewrites]
              [--kore-rpc-command KORE_RPC_COMMAND] [--use-booster]
              [--no-use-booster] [--port PORT]
              [--maude-port MAUDE_PORT] [--bug-report BUG_REPORT]
              [--break-every-step] [--break-on-jumpi]
              [--break-on-calls] [--no-break-on-calls]
              [--break-on-storage] [--break-on-basic-blocks]
              [--symbolic-immutables] [--max-depth MAX_DEPTH]
              [--max-iterations MAX_ITERATIONS] [--failure-information]
              [--no-failure-information] [--auto-abstract-gas]
              [--counterexample-information]
              [--no-counterexample-information] [--fail-fast]
              [--force-sequential] [--no-fail-fast]
              [--foundry-project-root FOUNDRY_ROOT]
              [--enum-constraints]
              [--schedule {DEFAULT,FRONTIER,HOMESTEAD,TANGERINE_WHISTLE,SPURIOUS_
              [--chainid CHAINID] [--mode {NORMAL,VMTESTS}] [--no-gas]
              [--config-file CONFIG_FILE]
              [--config-profile CONFIG_PROFILE] [--match-test TESTS]
              [--reinit] [--setup-version SETUP_VERSION]
              [--max-frontier-parallel MAX_FRONTIER_PARALLEL]
              [--bmc-depth BMC_DEPTH] [--run-constructor] [--use-gas]
              [--config-type {ConfigType.TEST_CONFIG,ConfigType.SUMMARY_CONFIG}]
              [--hide-status-bar] [--break-on-cheatcodes]
              [--init-node-from-diff RECORDED_DIFF_STATE_PATH]
              [--init-node-from-dump RECORDED_DUMP_STATE_PATH]
              [--include-summary INCLUDE_SUMMARIES]
              [--with-non-general-state] [--xml-test-report] [--cse]
              [--hevm] [--minimize-proofs] [--evm-tracing]
              [--no-trace-storage] [--no-trace-wordstack]
              [--no-trace-memory] [--remove-old-proofs]
              [--optimize-performance OPTIMIZE_PERFORMANCE]
              [--no-stack-checks]
```

Kontrol prove: weaponry

runtime verification

Relevant options revolve around **proof execution**:

- Proof **execution tweaks**
  - Get more **detailed execution** trees
  - **Loop exploration** behavior
  - **Performance** increases

Kontrol prove: weaponry

runtime verification

Get more **detailed execution** trees

- Modulate when Kontrol creates KCFG nodes:
  - `--max-depth N` At most each `N` steps create a node
  - Make a node (break) on **special occasions**
    - `--break-every-step`: On every EVM opcode
    - `--break-on-jumpi`: On every JUMPI opcode
    - `--break-on-storage`: On every SSTORE/SLOAD opcode
    - `--break-on-cheatcodes`: On every cheatcode

Kontrol prove: weaponry

runtime verification

Loop exploration behavior

- By default Kontrol doesn't bound loop execution
- Unbounded loops become potentially infinite loops
- There are two ways of dealing with this
  - Bounded loop unrolling: `--bmc-depth N`
  - Specifying loop invariants

Kontrol prove: weaponry

runtime verification

Performance increases

As part of Kontrol 1.0, there is the comprehensive option

```
--optimize-performance N
```

- Overrides proof execution settings for maximum speed
- `N` is the number of branches to explore concurrently

Proof Inspection 🕵️

runtime verification

`kontrol prove` said ✨ **PROOF PASSED** ✨ or ❌ **PROOF FAILED** ❌, now what?

- `kontrol view-kcfg`: TUI interactive viewer
- `kontrol show`: CLI output
  - Also used for execution deltas (more on that later)
- KaaS: Web view and proof storage management service

Proof Inspection 🧐

runtime verification

Stuff you can do with your kontrol prove result

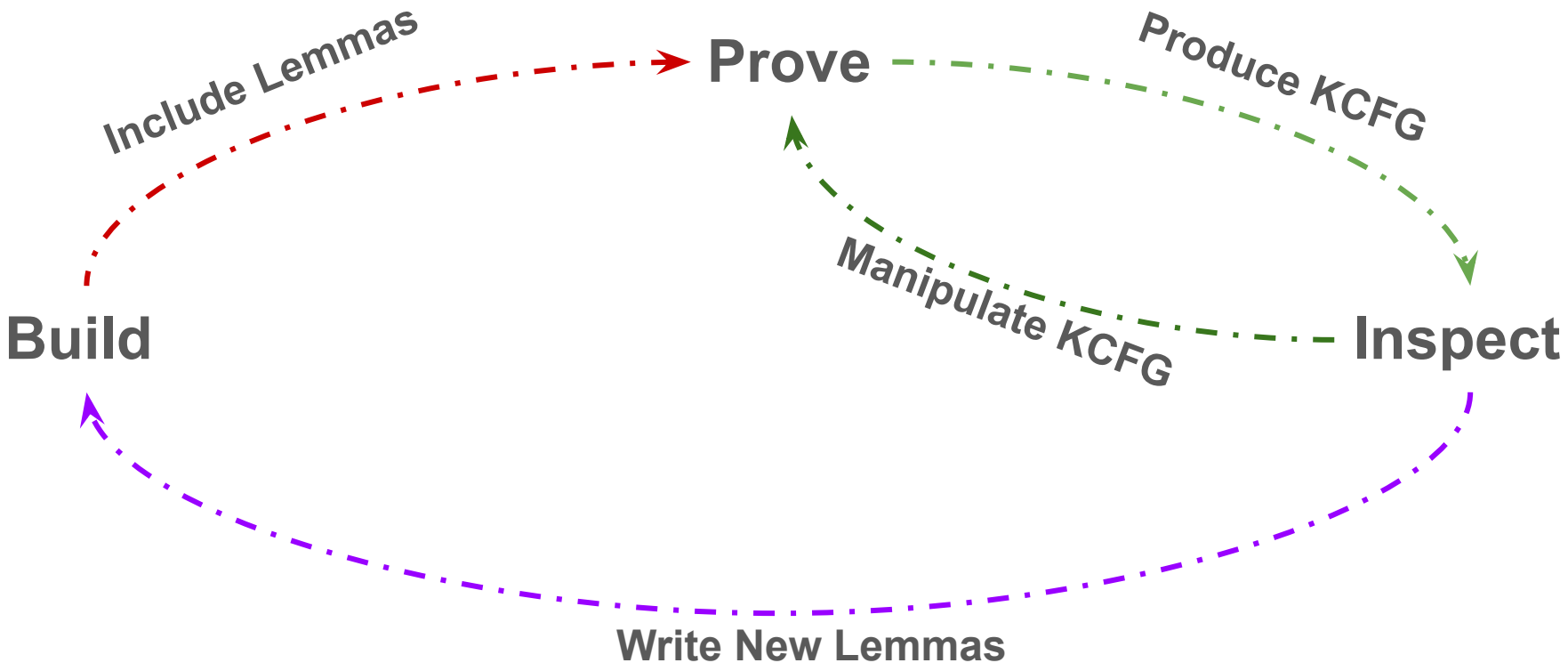
- KCFG **manipulation**:
  - `kontrol remove-node`: Remove a node and its successors
  - `kontrol refute-node`: Remove a branch from being explored
  - Many more…

Possible reaction to all of this

runtime verification

Kontrol Workflow Loop

runtime verification



runtime verification

Applying the Verification Cycle.

Our Working Example

runtime verification

```
function sumToN(uint256 n) public pure returns (uint256) {
    uint256 result = 0;
    uint256 i = 0;
    while (i < n) {
        i = i + 1;
        result = result + i;
    }
    return result;
}
```

Our Working Example

runtime verification

```
function sumToN(uint256 n) public pure returns (uint256) {
    uint256 result = 0;
    uint256 i = 0;
    while (i < n) {
        i = i + 1;
        result = result + i;
    }
    return result;
}
```

$$\sum_{i=0}^{n} i$$

Our Working Example

runtime verification

```
function sumToN(uint256 n) public pure returns (uint256) {
    uint256 result = 0;
    uint256 i = 0;
    while (i < n) {
        i = i + 1;
        result = result + i;
    }
    return result;
}
```

Our Working Example

runtime verification

```
function sumToN(uint256 n) public pure returns (uint256) {
    uint256 result = 0;
    uint256 i = 0;
    while (i < n) {
        i = i + 1;
        result = result + i;
    }
    return result;
}
```

Our Working Example

runtime verification

```
function sumToN(uint256 n) public pure returns (uint256) {
    uint256 result = 0;
    uint256 i = 0;
    while (i < n) {
        i = i + 1;
        result = result + i;
    }
    return result;
}
```

Our Working Example

runtime verification

```
function sumToN(uint256 n) public pure returns (uint256) {
    uint256 result = 0;
    uint256 i = 0;
    while (i < n) {
        i = i + 1;
        result = result + i;
    }
    return result;
}
```

Creating the Specs

runtime verification

`sumToN(uint256 n)` = $\sum_{i=0}^{n} i$

Creating the Specs

runtime verification

`sumToN(uint256 n)` = $\sum_{i=0}^{n} i$

$$\sum_{i=0}^{n} i = n * (n + 1) / 2$$

Creating the Specs

runtime verification

```
function prove_sumToN(uint256 n) external pure {
    vm.assume(n < 2**128); // prevent overflow
    assert(sumToN(n) == n * (n + 1) / 2);
}
```

Executing the Specs

runtime verification

```
function prove_sumToN(uint256 n) external pure {
    vm.assume(n < 2**128); // prevent overflow
    assert(sumToN(n) == n * (n + 1) / 2);
}
```

- `kontrol build`
- `kontrol prove --mt prove_sumToN`

Making the Proof Finish?

runtime verification

```
function prove_sumToN(uint256 n) external pure {
    vm.assume(n < 2**128); // prevent overflow
    assert(sumToN(n) == n * (n + 1) / 2);
}
```

- `kontrol build`
- `kontrol prove --mt prove_sumToN`

Making the Proof Finish

runtime verification

```
function prove_sumToN(uint256 n) external pure {
    vm.assume(n < 2**128); // prevent overflow
    assert(sumToN(n) == n * (n + 1) / 2);
}
```

- `kontrol build`
- `kontrol prove --mt prove_sumToN --bmc-depth 4`

Malicious `sumToN`

runtime verification

```
function maliciousSumToN(uint256 n) public pure returns (uint256) {
    uint256 result = 0;
    uint256 i = 0;
    while (i < n) {
        i = i + 1;
        result = result + i;
        if (i > 4) result = 42;
    }
    return result;
}
```

Malicious `sumToN`

runtime verification

```
function maliciousSumToN(uint256 n) public pure returns (uint256) {
    uint256 result = 0;
    uint256 i = 0;
    while (i < n) {
        i = i + 1;
        result = result + i;
        if (i > 4) result = 42;
    }
    return result;
}
```

Running the malicious sum

runtime verification

```
function prove_maliciousSumToN(uint256 n) external pure {
    vm.assume(n < 2**128); // prevent overflow
    assert(maliciousSumToN(n) == n * (n + 1) / 2);
}
```

- `kontrol build`
- `kontrol prove --mt prove_sumToN --bmc-depth 4`

Loop Invariants

runtime verification

- What is an invariant?
  - Properties that hold **before** and **after** executing some code
- What is a **loop** invariant?
  - Properties that hold **before** and **after** executing a **loop**

Loop Invariants: forging the intuition

runtime verification

```
function sumToN(uint256 n) public pure returns (uint256) {
    uint256 result = 0;
    uint256 i = 0;
    while (i < n) {
        i = i + 1;
        result = result + i;
    }
    return result;
}
```

Loop Invariants: forging the intuition

runtime verification

```
function sumToN(uint256 n) public pure returns (uint256) {
    uint256 result = 0;
    uint256 i = 0;
    while (i < n) {
        i = i + 1;
        result = result + i;
    }
    return result;
}
```

- Before entering the loop
  - `result = i = 0 = i * (i + 1) / 2`

Loop Invariants: forging the intuition

runtime verification

```
function sumToN(uint256 n) public pure returns (uint256) {
    uint256 result = 0;
    uint256 i = 0;
    while (i < n) {
        i = i + 1;
        result = result + i;
    }
    return result;
}
```

- Before entering the loop
  - `result = i = 0 = i * (i + 1) / 2`
- After executing the loop
  - `i = n`

Loop Invariants: forging the intuition

runtime verification

```
function sumToN(uint256 n) public pure returns (uint256) {
    uint256 result = 0;
    uint256 i = 0;
    while (i < n) {
        i = i + 1;
        result = result + i;
    }
    return result;
}
```

- Before entering the loop
  - `result = i = 0 = i * (i + 1) / 2`
- After executing the loop
  - `i = n`
  - `result = n * (n + 1) / 2 = i * (i + 1) / 2` (hopefully)

Loop Invariants: the Recipe

runtime verification

- Look at the loop head and see how the state is
- Execute the loop head and see how the state is updated
- Generalize the state update into a property

Loop Invariants: the Recipe for EVM

runtime verification

- Identify the loop head
- Look at the loop head and see how the state is
- Execute the loop head and see how the state is updated
- Generalize the state update into a property
- **Prove** that your invariant is correct
- Include the invariant as a rewrite step

A Word on Lemmas

runtime verification

```
requires "foundry.md"

module MY-LEMMAS
    imports BOOL
    imports FOUNDRY
    imports INT-SYMBOLIC

    rule bool2Word ( X ) => 1 requires X         [simplification]
    rule bool2Word ( X ) => 0 requires notBool X [simplification]

    rule chop ( bool2Word ( X ) <<Int Y ) => bool2Word ( X ) <<Int Y requires Y <Int 256 [simplification]

    rule chop ( 1 <<Int Y ) => 1 <<Int Y requires Y <Int 256 [simplification]

    rule ( A:Int >>Int B:Int ) >>Int C:Int => A >>Int ( B +Int C ) [simplification]

    rule A =/=Int B => notBool ( A ==Int B ) [simplification, comm]

endmodule
```

A Word on Lemmas

runtime verification

```
requires "foundry.md"

module MY-LEMMAS
    imports BOOL
    imports FOUNDRY
    imports INT-SYMBOLIC

    rule bool2Word ( X ) => 1 requires X         [simplification]
    rule bool2Word ( X ) => 0 requires notBool X [simplification]

    rule chop ( bool2Word ( X ) <<Int Y ) => bool2Word ( X ) <<Int Y requires Y <Int 256 [simplification]

    rule chop ( 1 <<Int Y ) => 1 <<Int Y requires Y <Int 256 [simplification]

    rule ( A:Int >>Int B:Int ) >>Int C:Int => A >>Int ( B +Int C ) [simplification]

    rule A =/=Int B => notBool ( A ==Int B ) [simplification, comm]

endmodule
```

A Word on Lemmas

runtime verification

```
requires "foundry.md"

module MY-LEMMAS
    imports BOOL
    imports FOUNDRY
    imports INT-SYMBOLIC

    rule bool2Word ( X ) => 1 requires X         [simplification]
    rule bool2Word ( X ) => 0 requires notBool X [simplification]

    rule chop ( bool2Word ( X ) <<Int Y ) => bool2Word ( X ) <<Int Y requires Y <Int 256 [simplification]

    rule chop ( 1 <<Int Y ) => 1 <<Int Y requires Y <Int 256 [simplification]

    rule ( A:Int >>Int B:Int ) >>Int C:Int => A >>Int ( B +Int C ) [simplification]

    rule A =/=Int B => notBool ( A ==Int B ) [simplification, comm]

endmodule
```

A Word on Lemmas

runtime verification

```
requires "foundry.md"

module MY-LEMMAS
    imports BOOL
    imports FOUNDRY
    imports INT-SYMBOLIC

    rule bool2Word ( X ) => 1 requires X         [simplification]
    rule bool2Word ( X ) => 0 requires notBool X [simplification]

    rule chop ( bool2Word ( X ) <<Int Y ) => bool2Word ( X ) <<Int Y requires Y <Int 256 [simplification]

    rule chop ( 1 <<Int Y ) => 1 <<Int Y requires Y <Int 256 [simplification]

    rule ( A:Int >>Int B:Int ) >>Int C:Int => A >>Int ( B +Int C ) [simplification]

    rule A =/=Int B => notBool ( A ==Int B ) [simplification, comm]

endmodule
```

A Word on Lemmas

runtime verification

```
requires "foundry.md"

module MY-LEMMAS
    imports BOOL
    imports FOUNDRY
    imports INT-SYMBOLIC

    rule bool2Word ( X ) => 1 requires X         [simplification]
    rule bool2Word ( X ) => 0 requires notBool X [simplification]

    rule chop ( bool2Word ( X ) <<Int Y ) => bool2Word ( X ) <<Int Y requires Y <Int 256 [simplification]

    rule chop ( 1 <<Int Y ) => 1 <<Int Y requires Y <Int 256 [simplification]

    rule ( A:Int >>Int B:Int ) >>Int C:Int => A >>Int ( B +Int C ) [simplification]

    rule A =/=Int B => notBool ( A ==Int B ) [simplification, comm]

endmodule
```

A Word on Lemmas

runtime verification

```
requires "foundry.md"

module MY-LEMMAS
    imports BOOL
    imports FOUNDRY
    imports INT-SYMBOLIC

    rule bool2Word ( X ) => 1 requires X         [simplification]
    rule bool2Word ( X ) => 0 requires notBool X [simplification]

    rule chop ( bool2Word ( X ) <<Int Y ) => bool2Word ( X ) <<Int Y requires Y <Int 256 [simplification]

    rule chop ( 1 <<Int Y ) => 1 <<Int Y requires Y <Int 256 [simplification]

    rule ( A:Int >>Int B:Int ) >>Int C:Int => A >>Int ( B +Int C ) [simplification]

    rule A =/=Int B => notBool ( A ==Int B ) [simplification, comm]

endmodule
```

A Word on Lemmas

runtime verification

```
requires "foundry.md"

module MY-LEMMAS
    imports BOOL
    imports FOUNDRY
    imports INT-SYMBOLIC

    rule bool2Word ( X ) => 1 requires X         [simplification]
    rule bool2Word ( X ) => 0 requires notBool X [simplification]

    rule chop ( bool2Word ( X ) <<Int Y ) => bool2Word ( X ) <<Int Y requires Y <Int 256 [simplification]

    rule chop ( 1 <<Int Y ) => 1 <<Int Y requires Y <Int 256 [simplification]

    rule ( A:Int >>Int B:Int ) >>Int C:Int => A >>Int ( B +Int C ) [simplification]

    rule A =/=Int B => notBool ( A ==Int B ) [simplification, comm]

endmodule
```

A Word on Lemmas

runtime verification

```
requires "foundry.md"

module MY-LEMMAS
    imports BOOL
    imports FOUNDRY
    imports INT-SYMBOLIC

    rule bool2Word ( X ) => 1 requires X         [simplification]
    rule bool2Word ( X ) => 0 requires notBool X [simplification]

    rule chop ( bool2Word ( X ) <<Int Y ) => bool2Word ( X ) <<Int Y requires Y <Int 256 [simplification]

    rule chop ( 1 <<Int Y ) => 1 <<Int Y requires Y <Int 256 [simplification]

    rule ( A:Int >>Int B:Int ) >>Int C:Int => A >>Int ( B +Int C ) [simplification]

    rule A =/=Int B => notBool ( A ==Int B ) [simplification, comm]

endmodule
```

A Word on Lemmas

runtime verification

```
requires "foundry.md"

module MY-LEMMAS
    imports BOOL
    imports FOUNDRY
    imports INT-SYMBOLIC

    rule bool2Word ( X ) => 1 requires X         [simplification]
    rule bool2Word ( X ) => 0 requires notBool X [simplification]

    rule chop ( bool2Word ( X ) <<Int Y ) => bool2Word ( X ) <<Int Y requires Y <Int 256 [simplification]

    rule chop ( 1 <<Int Y ) => 1 <<Int Y requires Y <Int 256 [simplification]

    rule ( A:Int >>Int B:Int ) >>Int C:Int => A >>Int ( B +Int C ) [simplification]

    rule A =/=Int B => notBool ( A ==Int B ) [simplification, comm]

endmodule
```

A Word on Lemmas

runtime verification

```
requires "foundry.md"

module MY-LEMMAS
    imports BOOL
    imports FOUNDRY
    imports INT-SYMBOLIC

    rule bool2Word ( X ) => 1 requires X          [simplification]
    rule bool2Word ( X ) => 0 requires notBool X [simplification]

    rule chop ( bool2Word ( X ) <<Int Y ) => bool2Word ( X ) <<Int Y requires Y <Int 256 [simplification]

    rule chop ( 1 <<Int Y ) => 1 <<Int Y requires Y <Int 256 [simplification]

    rule ( A:Int >>Int B:Int ) >>Int C:Int => A >>Int ( B +Int C ) [simplification]

    rule A =/=Int B => notBool ( A ==Int B ) [simplification, comm]

endmodule
```

A Word on Lemmas

runtime verification

```
requires "foundry.md"

module MY-LEMMAS
    imports BOOL
    imports FOUNDRY
    imports INT-SYMBOLIC

    rule bool2Word ( X ) => 1 requires X        [simplification]
    rule bool2Word ( X ) => 0 requires notBool X [simplification]

    rule chop ( bool2Word ( X ) <<Int Y ) => bool2Word ( X ) <<Int Y requires Y <Int 256 [simplification]

    rule chop ( 1 <<Int Y ) => 1 <<Int Y requires Y <Int 256 [simplification]

    rule ( A:Int >>Int B:Int ) >>Int C:Int => A >>Int ( B +Int C ) [simplification]

    rule A =/=Int B => notBool ( A ==Int B ) [simplification, comm]

endmodule
```

Proving the Invariant

runtime verification

```
claim [gauss-claim]:
  <k>
   ( JUMPI 775 bool2Word ( N:Int <=Int I:Int ) => JUMP 775 )
   ~> #pc [ JUMPI ]
   ~> #execute
   ~> _CONTINUATION:K
  </k>
  <useGas>
    false
  </useGas>
  <program>
    #binRuntime
  </program>
  <jumpDests>
    #computeValidJumpDests ( #binRuntime )
  </jumpDests>
  <wordStack>
    ( I => N ) : ( I *Int (I +Int 1) /Int 2 => N *Int (N +Int 1) /Int 2 ) : 0 : N : WS:WordStack
  </wordStack>
  <pc>
    745
  </pc>
  <activeTracing>
    false
  </activeTracing>
  <stackChecks>
    true
  </stackChecks>
  requires 0 <=Int N andBool N <Int 2 ^Int 128
   andBool 0 <=Int I andBool I <=Int N
   andBool #sizeWordStack(WS) <Int 1013
```

Proving the Invariant

runtime verification

```
claim [gauss-claim]:
  <k>
   ( JUMPI 775 bool2Word ( N:Int <=Int I:Int ) => JUMP 775 )
   ~> #pc [ JUMPI ]
   ~> #execute
   ~> _CONTINUATION:K
  </k>
  <useGas>
    false
  </useGas>
  <program>
    #binRuntime
  </program>
  <jumpDests>
    #computeValidJumpDests ( #binRuntime )
  </jumpDests>
  <wordStack>
    ( I => N ) : ( I *Int (I +Int 1) /Int 2 => N *Int (N +Int 1) /Int 2 ) : 0 : N : WS:WordStack
  </wordStack>
  <pc>
    745
  </pc>
  <activeTracing>
    false
  </activeTracing>
  <stackChecks>
    true
  </stackChecks>
  requires 0 <=Int N andBool N <Int 2 ^Int 128
   andBool 0 <=Int I andBool I <=Int N
   andBool #sizeWordStack(WS) <Int 1013
```

Proving the Invariant

runtime verification

```
claim [gauss-claim]:
 <k>
  ( JUMPI 775 bool2Word ( N:Int <=Int I:Int ) => JUMP 775 )
  ~> #pc [ JUMPI ]
  ~> #execute
  ~> _CONTINUATION:K
 </k>
 <useGas>
   false
 </useGas>
 <program>
   #binRuntime
 </program>
 <jumpDests>
   #computeValidJumpDests ( #binRuntime )
 </jumpDests>
 <wordStack>
   ( I => N ) : ( I *Int (I +Int 1) /Int 2 => N *Int (N +Int 1) /Int 2 ) : 0 : N : WS:WordStack
 </wordStack>
 <pc>
   745
 </pc>
 <activeTracing>
   false
 </activeTracing>
 <stackChecks>
   true
 </stackChecks>
 requires 0 <=Int N andBool N <Int 2 ^Int 128
  andBool 0 <=Int I andBool I <=Int N
  andBool #sizeWordStack(WS) <Int 1013
```

Proving the Invariant

runtime verification

```
claim [gauss-claim]:
  <k>
   ( JUMPI 775 bool2Word ( N:Int <=Int I:Int ) => JUMP 775 )
   ~> #pc [ JUMPI ]
   ~> #execute
   ~> _CONTINUATION:K
  </k>
  <useGas>
    false
  </useGas>
  <program>
    #binRuntime
  </program>
  <jumpDests>
    #computeValidJumpDests ( #binRuntime )
  </jumpDests>
  <wordStack>
    ( I => N ) : ( I *Int (I +Int 1) /Int 2 => N *Int (N +Int 1) /Int 2 ) : 0 : N : WS:WordStack
  </wordStack>
  <pc>
    745
  </pc>
  <activeTracing>
    false
  </activeTracing>
  <stackChecks>
    true
  </stackChecks>
  requires 0 <=Int N andBool N <Int 2 ^Int 128
   andBool 0 <=Int I andBool I <=Int N
   andBool #sizeWordStack(WS) <Int 1013
```

Proving the Invariant

runtime verification

```
claim [gauss-claim]:
  <k>
   ( JUMPI 775 bool2Word ( N:Int <=Int I:Int ) => JUMP 775 )
   ~> #pc [ JUMPI ]
   ~> #execute
   ~> _CONTINUATION:K
  </k>
  <useGas>
    false
  </useGas>
  <program>
    #binRuntime
  </program>
  <jumpDests>
    #computeValidJumpDests ( #binRuntime )
  </jumpDests>
  <wordStack>
    ( I => N ) : ( I *Int (I +Int 1) /Int 2 => N *Int (N +Int 1) /Int 2 ) : 0 : N : WS:WordStack
  </wordStack>
  <pc>
    745
  </pc>
  <activeTracing>
    false
  </activeTracing>
  <stackChecks>
    true
  </stackChecks>
  requires 0 <=Int N andBool N <Int 2 ^Int 128
   andBool 0 <=Int I andBool I <=Int N
   andBool #sizeWordStack(WS) <Int 1013
```

Proving the Invariant

runtime verification

```
claim [gauss-claim]:
  <k>
   ( JUMPI 775 bool2Word ( N:Int <=Int I:Int ) => JUMP 775 )
   ~> #pc [ JUMPI ]
   ~> #execute
   ~> _CONTINUATION:K
  </k>
  <useGas>
    false
  </useGas>
  <program>
    #binRuntime
  </program>
  <jumpDests>
    #computeValidJumpDests ( #binRuntime )
  </jumpDests>
  <wordStack>
    ( I => N ) : ( I *Int (I +Int 1) /Int 2 => N *Int (N +Int 1) /Int 2 ) : 0 : N : WS:WordStack
  </wordStack>
  <pc>
    745
  </pc>
  <activeTracing>
    false
  </activeTracing>
  <stackChecks>
    true
  </stackChecks>
  requires 0 <=Int N andBool N <Int 2 ^Int 128
   andBool 0 <=Int I andBool I <=Int N
   andBool #sizeWordStack(WS) <Int 1013
```

Proving the Invariant

runtime verification

```
claim [gauss-claim]:
  <k>
   ( JUMPI 775 bool2Word ( N:Int <=Int I:Int ) => JUMP 775 )
   ~> #pc [ JUMPI ]
   ~> #execute
   ~> _CONTINUATION:K
  </k>
  <useGas>
    false
  </useGas>
  <program>
    #binRuntime
  </program>
  <jumpDests>
    #computeValidJumpDests ( #binRuntime )
  </jumpDests>
  <wordStack>
    ( I => N ) : ( I *Int (I +Int 1) /Int 2 => N *Int (N +Int 1) /Int 2 ) : 0 : N : WS:WordStack
  </wordStack>
  <pc>
    745
  </pc>
  <activeTracing>
    false
  </activeTracing>
  <stackChecks>
    true
  </stackChecks>
  requires 0 <=Int N andBool N <Int 2 ^Int 128
   andBool 0 <=Int I andBool I <=Int N
   andBool #sizeWordStack(WS) <Int 1013
```

Proving the Invariant

runtime verification

```
claim [gauss-claim]:
  <k>
   ( JUMPI 775 bool2Word ( N:Int <=Int I:Int ) => JUMP 775 )
   ~> #pc [ JUMPI ]
   ~> #execute
   ~> _CONTINUATION:K
  </k>
  <useGas>
    false
  </useGas>
  <program>
    #binRuntime
  </program>
  <jumpDests>
    #computeValidJumpDests ( #binRuntime )
  </jumpDests>
  <wordStack>
    ( I => N ) : ( I *Int (I +Int 1) /Int 2 => N *Int (N +Int 1) /Int 2 ) : 0 : N : WS:WordStack
  </wordStack>
  <pc>
    745
  </pc>
  <activeTracing>
    false
  </activeTracing>
  <stackChecks>
    true
  </stackChecks>
 requires 0 <=Int N andBool N <Int 2 ^Int 128
  andBool 0 <=Int I andBool I <=Int N
  andBool #sizeWordStack(WS) <Int 1013
```

Turning the Claim into a Lemma

runtime verification

```
rule [gauss-invariant]:
  <k>
   ( JUMPI 685 CONDITION => JUMP 685 )
   ~> #pc [ JUMPI ]
   ~> #execute
   ~> _CONTINUATION:K
  </k>
  <useGas>
    false
  </useGas>
  <program>
    PROGRAM
  </program>
  <jumpDests>
    JUMPDESTS
  </jumpDests>
  <wordStack>
    ( I => N ) : ( RESULT => N *Int (N +Int 1) /Int 2 ) : 0 : N : WS:WordStack
  </wordStack>
  <pc>
    745
  </pc>
  <activeTracing>
    false
  </activeTracing>
  requires CONDITION ==K bool2Word ( N:Int <=Int I:Int )
   andBool PROGRAM   ==K #binRuntime
   andBool JUMPDESTS ==K #computeValidJumpDests ( #binRuntime )
   andBool RESULT    ==Int I *Int (I +Int 1) /Int 2
   andBool 0 <=Int N andBool N <Int 2 ^Int 128
   andBool 0 <=Int I andBool I <=Int N
   andBool #sizeWordStack(WS) <Int 1013
  [priority(30)]
```

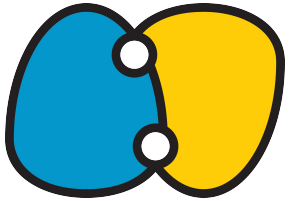
Get excited for Kontrol!

runtime verification

- Open source and free to use!
- Used by major projects independently and with our help
- Compatible with Foundry
- Can be run in CI
- Kontrol 1.0 is the most performant!

github.com/runtimeverification/install-kontrol

runtime verification



Stay in touch!

@rv_inc

@BatisteFormal

https://discord.com/invite/CurfmXNtbN

https://docs.runtimeverification.com/kontrol

https://github.com/runtimeverification/kontrol

https://t.me/rv_kontrol