Horus: A Formal Verification Tool for Starknet

Julian Sutherland

RV/ILDS blockchain workshop, Bucharest, Romania

What is Starknet?

- Ethereum L2.

What is Starknet?

- Ethereum L2.
- ZK-Rollup using STARKs.

What is Starknet?

- Ethereum L2.
- ZK-Rollup using STARKs.
- "Unusual" bytecode used to make proof generation and verification as efficient as possible: Cairo.

What is Starknet?

- Ethereum L2.
- ZK-Rollup using STARKs.
- “Unusual” bytecode used to make proof generation and verification as efficient as possible: Cairo.
- The Warp compiler, also developed at Nethermind, can compile Solidity into Cairo.

A quick overview of the Cairo bytecode.

- Runs on a non-deterministic stack machine, with a small instruction set: Assertions, Calls, Returns, Jumps and Incrementing allocation pointer (ap).
- Memory is read-only (or write-once).
- Primitive values are *felts*: $\mathbb{F}_{2^{251}+17\times 2^{192}+1}$.
- Builtins:
  - Syscall.
    - Storage memory read/write.
    - `get_block_timestamp`
    - `get_caller_address`
    - `get_contract_address`
    - ...
  - Range check.
  - ...

Examples

```
@known_ap_change
func is_le_felt{range_check_ptr}(a : felt, b : felt) -> felt {
  %{ memory[ap] = 0 if (ids.a % PRIME) <= (ids.b % PRIME) else 1 %}
  jmp not_le if [ap] != 0, ap++;
  ap += 6;
  assert_le_felt(a, b);
  return 1;

  not_le:
  assert_lt_felt(b, a);
  return 0;
}
```

```
f();
let (x) = 0;
f();
```

Examples

```
@known_ap_change
func is_le_felt{range_check_ptr}(a : felt, b : felt) -> felt {
  %{ memory[ap] = 0 if (ids.a % PRIME) <= (ids.b % PRIME) else 1 %}
  jmp not_le if [ap] != 0, ap++;
  ap += 6;
  assert_le_felt(a, b);
  return 1;

  not_le:
  assert_lt_felt(b, a);
  return 0;
}
```

```
f();
let (x) = 0;
f();
```

Examples

```
@known_ap_change
func is_le_felt{range_check_ptr}(a : felt, b : felt) -> felt {
  %{ memory[ap] = 0 if (ids.a % PRIME) <= (ids.b % PRIME) else 1 %}
  jmp not_le if [ap] != 0, ap++;
  ap += 6;
  assert_le_felt(a, b);
  return 1;

  not_le:
  assert_lt_felt(b, a);
  return 0;
}
```

```
f();
let (x) = 0;
f();
```

Builtin examples

```
func assert_a_le_RCB{range_check_ptr}(a : felt) {
    [range_check_ptr] = a;
    let range_check_ptr = range_check_ptr + 1;
    return ();
}
```

```
func modify_account_balance{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*, range_check_ptr}
(account_id: felt, token_type: felt, amount: felt) {
  let (current_balance) = account_balance.read(account_id, token_type);
  tempvar new_balance = current_balance + amount;
  assert_nn_le(new_balance, BALANCE_UPPER_BOUND - 1);
  account_balance.write(account_id=account_id, token_type=token_type, value=new_balance);
  return ();
}
```

Builtin examples

```
func assert_a_le_RCB{range_check_ptr}(a : felt) {
    [range_check_ptr] = a;
    let range_check_ptr = range_check_ptr + 1;
    return ();
}
```

```
func modify_account_balance{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*, range_check_ptr}
(account_id: felt, token_type: felt, amount: felt) {
  let (current_balance) = account_balance.read(account_id, token_type);
  tempvar new_balance = current_balance + amount;
  assert_nn_le(new_balance, BALANCE_UPPER_BOUND - 1);
  account_balance.write(account_id=account_id, token_type=token_type, value=new_balance);
  return ();
}
```

Builtin examples

```
func assert_a_le_RCB{range_check_ptr}(a : felt) {
    [range_check_ptr] = a;
    let range_check_ptr = range_check_ptr + 1;
    return ();
}
```

```
func modify_account_balance{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*, range_check_ptr}
(account_id: felt, token_type: felt, amount: felt) {
  let (current_balance) = account_balance.read(account_id, token_type);
  tempvar new_balance = current_balance + amount;
  assert_nn_le(new_balance, BALANCE_UPPER_BOUND - 1);
  account_balance.write(account_id=account_id, token_type=token_type, value=new_balance);
  return ();
}
```

An introduction to Horus

- Horus is a formal verification tool allowing developers to annotate their smart contracts with partial *Hoare* logic specifications:

```
func get_opposite_token(token : felt) -> (t : felt) {
  if (token == 0) {
    return (t=1);
  } else {
    return (t=0);
  }
}
```

An introduction to Horus

- Horus is a formal verification tool allowing developers to annotate their smart contracts with partial *Hoare* logic specifications:

```
// @pre token == 0 or token == 1
// @post (token == 0 and $Return.t == 1) or (token == 1 and $Return.t == 0)
func get_opposite_token(token : felt) -> (t : felt) {
  if (token == 0) {
    return (t=1);
  } else {
    return (t=0);
  }
}
```

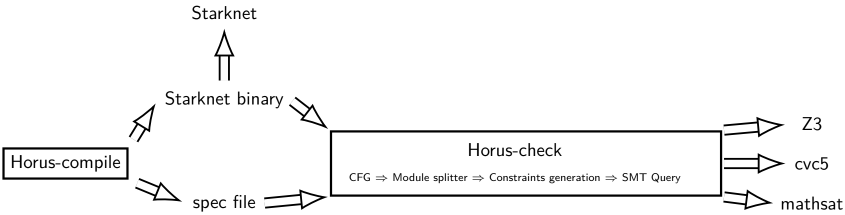
An introduction to Horus

- Horus is a formal verification tool allowing developers to annotate their smart contracts with partial *Hoare* logic specifications:

```
// @pre token == 0 or token == 1
// @post (token == 0 and $Return.t == 1) or (token == 1 and $Return.t == 0)
func get_opposite_token(token : felt) -> (t : felt) {
  if (token == 0) {
    return (t=1);
  } else {
    return (t=0);
  }
}
```

- Horus can then convert the problem of verifying that the implementations satisfy these specifications into SMT queries and discharged to appropriate solvers:
  - `horus-compile` can then generate a compiled Starknet binary and a specification file.
  - `horus-check` then generates queries and discharges them to a variety of SMT solvers, to verify that the generated binary satisfies the given specifications.

Horus architectural overview



A little more detail

- CFG generation:
  - Standard specs injected.
  - Inlining.
  - Optimising edges.
- Module splitting.
- Constraints generated for each module, substituting memory accesses.
- `NIA` SMT queries generated (in fact `QF_NIA` most of the time).
- Queries optimised using Z3 tactics.
- Queries discharged to SMT solvers.

Conclusions

- Horus is a powerful automated formal verification tool, which deals unusually well with non-linear specifications. Key examples:
  - toy AMM.
  - (simplified) `frob` function from Maker vat contract.
- Horus is has a relatively low skill ceiling.
- Next steps:
  - Separate specification file.
  - primitive `Int256`/`UInt256` support.
  - Cairo 1.0 update.
- Moon shots:
  - Other backends?
  - Osiris property tester.

Questions?

Try it yourself: https://github.com/NethermindEth/horus-checker

HORUS

NETHERMIND

Keep in touch:

@nethermindeth/@JulekSU
julian@nethermind.io