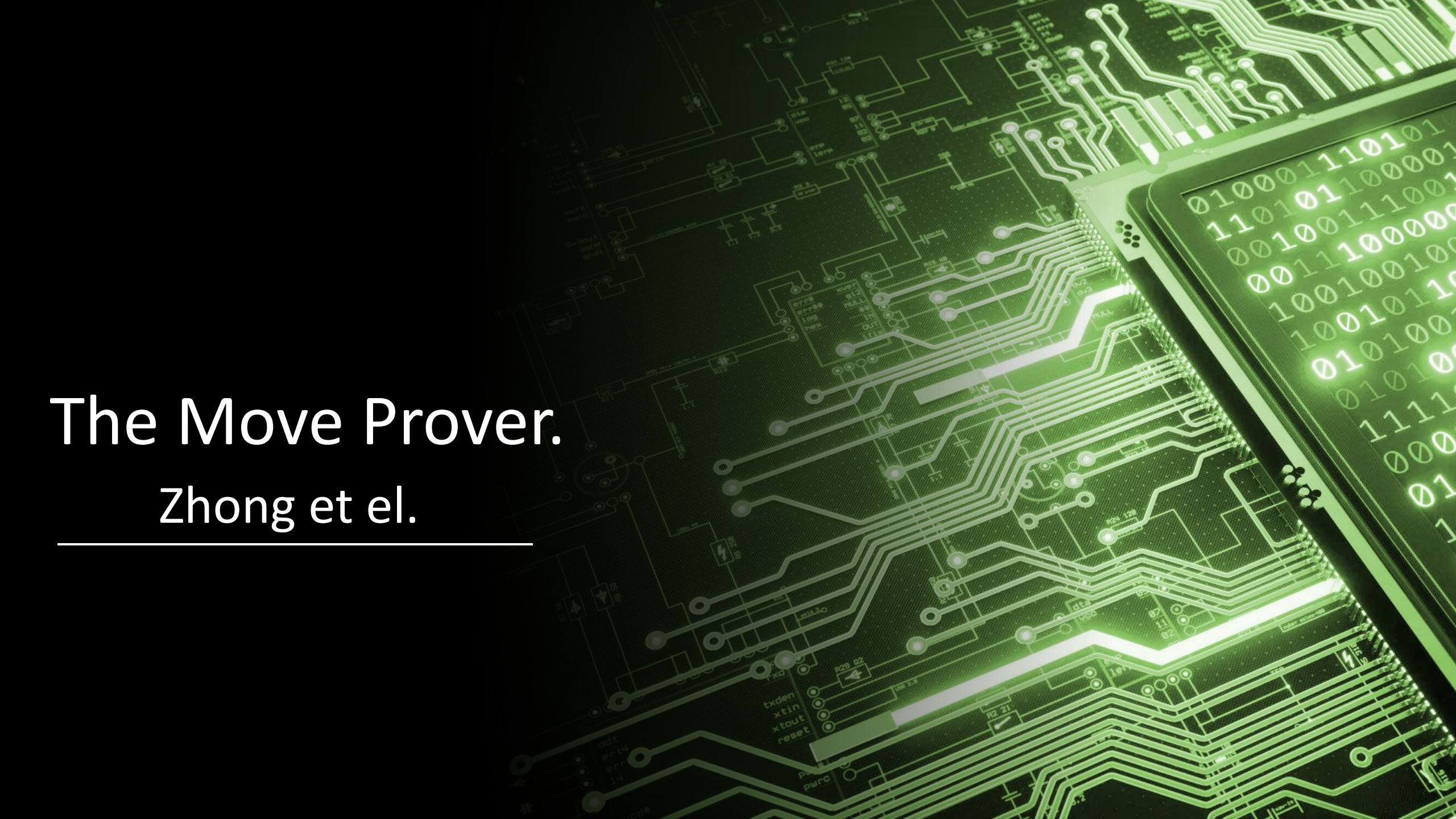
The Move Prover.

Zhong et el.



Agenda

- **Motivation**
- **Move Language Overview**
- **Move Prover Toolchain Implementation**
  - **Move Specification Language**
  - **Boogie Integration**
  - **Architecture & Flow**
- **Move Prover Evaluation & A Live Demo**

Motivation

Motivation

- The Libra blockchain is designed to store billions of dollars in assets.
- Bugs in smart contracts have led to massive amount of funds being stolen or made inaccessible.
- Move supports verification.
- Creating a culture of specification from the beginning.

An Example of a Move Resource:

Motivation

```
1  module M {
2      resource struct Counter {
3          value: u8,
4      }
5
6      public fun increment(a: address) acquires Counter {
7          let r = borrow_global_mut<Counter>(a);
8          r.value = r.value + 1;
9      }
```

Motivation

The Move Prover in Action

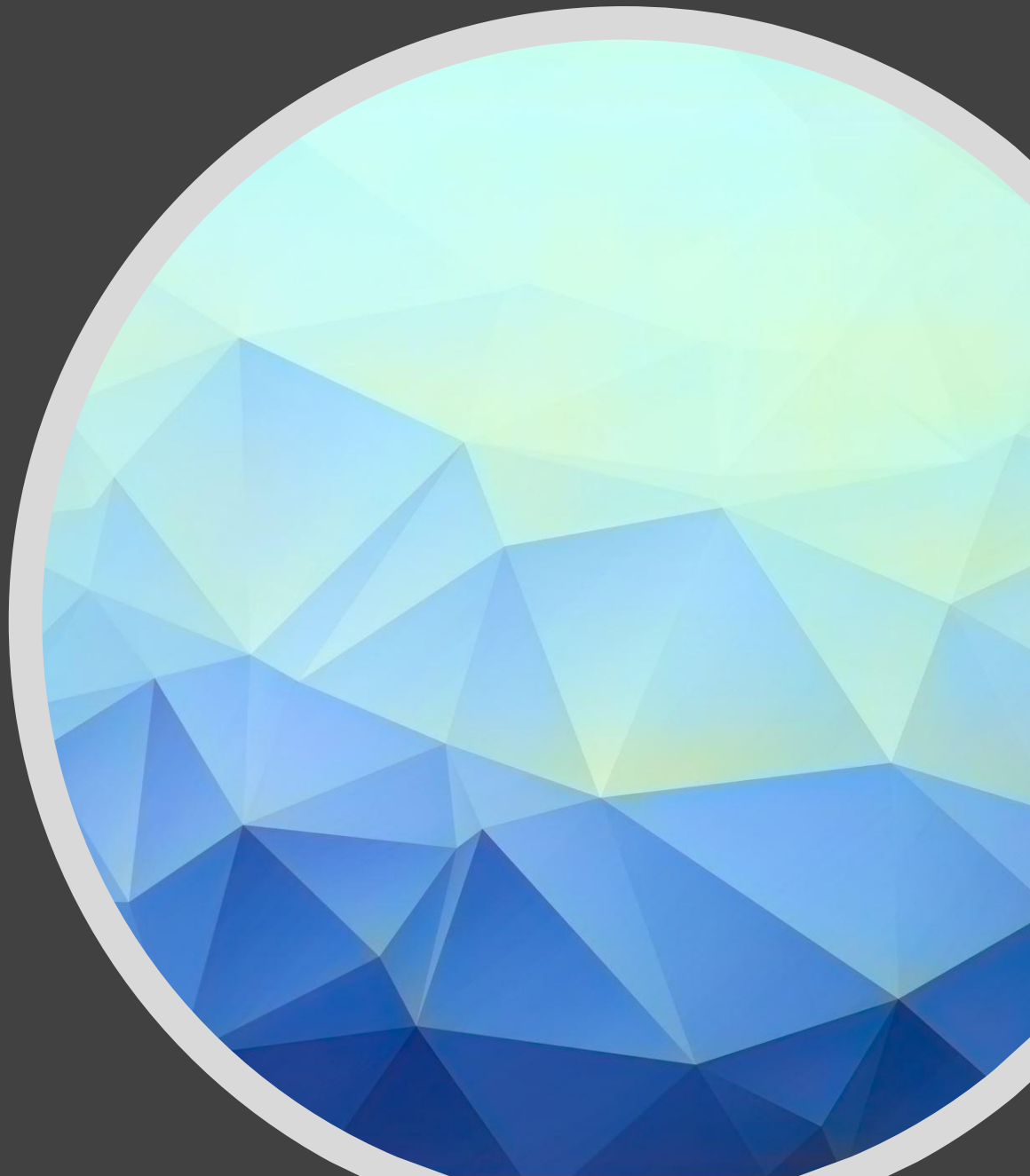
```
    ╭── tutorial.move:6:3 ──
    │
6   │ ╭     public fun increment(a: address) acquires Counter {
7   │ │         let r = borrow_global_mut<Counter>(a);
8   │ │         r.value = r.value + 1;
9   │ │     }
    │ ╰─────^
    ·
8   │           r.value = r.value + 1;
    │                              - abort happened here
    │
  =     at tutorial.move:6:3: increment (entry)
  =     at tutorial.move:7:15: increment
  =         a = 0x5,
  =         r = &M.Counter{value = 255u8}
  =     at tutorial.move:8:17: increment (ABORTED)
```

The Move Language in verification perspective

Overview

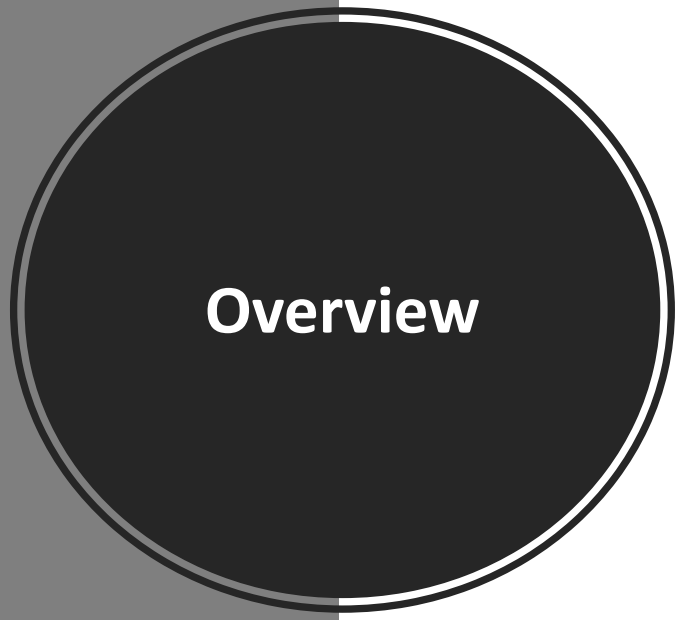
Features

Move bytecode verifier



Move Language

(Based on Shani Cattan slides)



- Blockchain high-level programming language for the Diem (Libra) Blockchain.
- Used to implement custom transactions and smart contracts.
- Developed by Meta (Facebook), 2019.
- Bytecode based, and has a bytecode verifier.

Features

The Move language is **verification friendly**:

- Depends only on global transaction & current transaction to ensure deterministic execution
- Any non deterministic operation is absent which causes challenges in verification.
- Safe arithmetic operations are set so verificator can rely on it.

Move bytecode verifier

Many common issues are ensured by the Move bytecode verifier, for example:

- Module dependencies are acyclic.
- A procedure can only touch stack locations belonging to callers via a reference passed to the callee.
- There are no dangling references.

And many more!

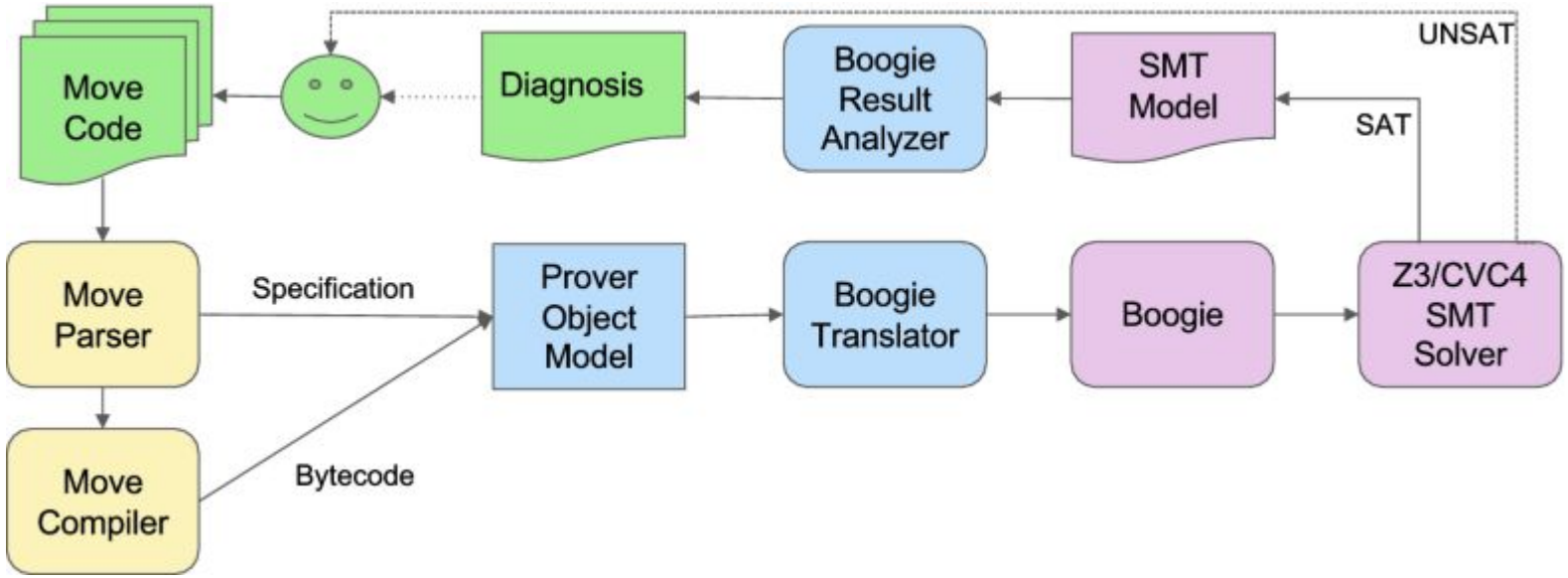
```
module LibraCoin {
  resource struct T { value: u64 }

  public fun join(coin: &mut LibraCoin::T, to_consume: LibraCoin::T) {
    let  T { value } = to_consume; // MoveLoc(1); Unpack
    let c_value_ref = &mut coin.value; // MoveLoc(0); MutBorrowField<value>; StLoc(0)
    *c_value_ref = *c_value_ref + value; // CopyLoc(0); ReadRef; Add; MoveLoc(0); WriteRef
    return; // Ret
  }
}
```

Move Prover Toolchain Implementation

Architecture

Move Prover Architecture

Move Prover Architecture



Floyd-Hoare

Specifications

- Formal reasoning about program correctness using pre- and postconditions.
- Syntax: {P} S {Q}. **P** & **Q** are predicates, **S** is the program.

{ true } x := 5 { x = 5 }

{ x = y } x := x + 3 { x = y + 3 }

{ x > -1 } x := x * 2 + 3 { x > 1 }

{ x < 0 } while (x != 0) x := x-1 {?}

Specifications

- Specifications include classical Floyd-Hoare pre-conditions, post-conditions, and a new condition specifying when a function aborts.
- Conditions are separated from the actual code.
- Specifications never affect the execution of a module.

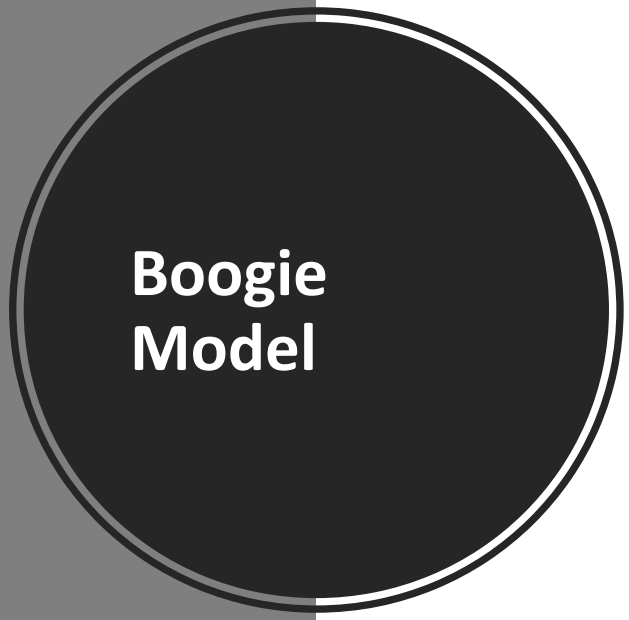
Specifications Syntax

- requires P, ensures P
  - old
- aborts if P
- exists<M::T>(A)
- global<M::T>(A)

Specifications


```
public fun pay_from_sender(payee: address, amount: u64) acquires T
{
  Transaction::assert(payee != Transaction::sender(), 1);  // new!

  if (!exists<T>(payee)) {
    Self::create_account(payee);
  };
  Self::deposit(
    payee,
    Self::withdraw_from_sender(amount),
  );
}

spec fun pay_from_sender {
//  ... omitted aborts_ifs ...
  aborts_if amount == 0;
  aborts_if global<T>(sender()).balance.value < amount;
  ensures exists<T>(payee);
  ensures global<T>(sender()).balance.value
      == old(global<T>(sender()).balance.value) - amount;
}
```

Boogie IVL



- Designed for verification.
- Used as backend for multiple verification tools.
- Gets a bpl file and translates it to SMT formulas

Evaluation

Statistics

Model Evaluation

Statistics

- The Libra and LibraAccount modules comprise nearly 1300 lines (including specifications). The total size of the generated Boogie files is a little over 14,000 lines, and the generated SMT files are around 52,000 lines

- The prover is used in continuous integration, and is beginning to be used to verify contracts in production

Move Module	LoC	Boogie LoC	SMT LoC	Functions	Verified	Runtime
Libra	420	3875	11,688	25	25	2.99 s
LibraAccount	867	10,362	40293	38	34	46.66 s

Live Demo

Questions?

Thank you for listening!