

Formal Specification and Verification of Solidity Contracts with Events

Ákos Hajdu¹, Dejan Jovanović², Gabriela Ciocarlie²

¹*Budapest University of Technology and Economics*

²*SRI International*

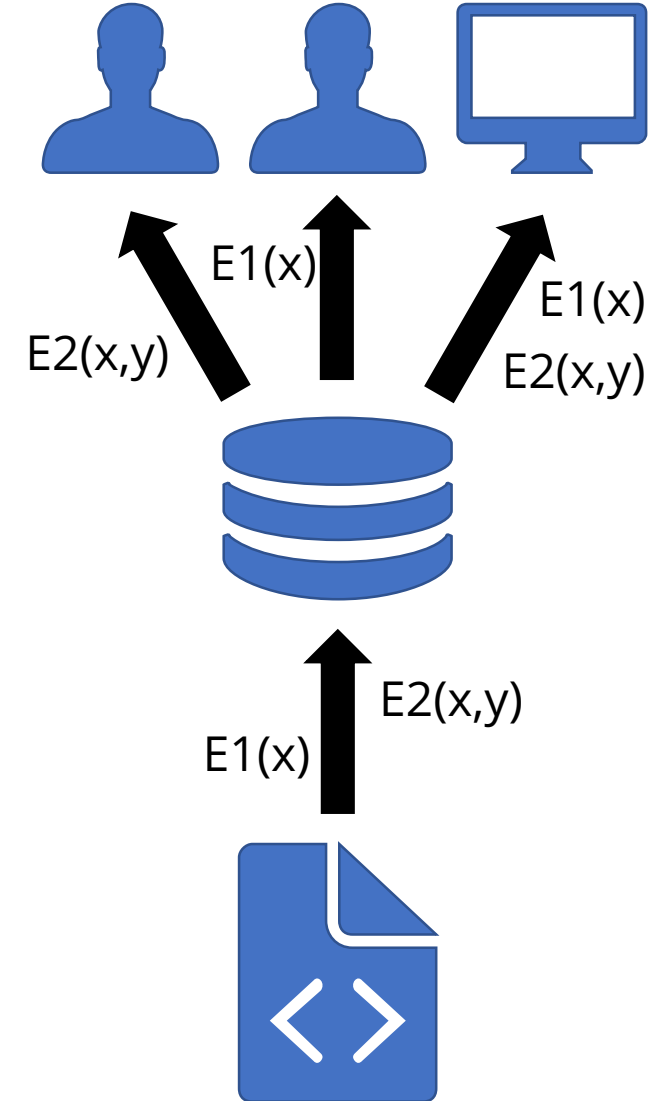


Solidity Smart Contracts and Events

```
contract Token {  
    mapping(address=>uint) balances;  
    uint total;  
  
    event initialized(address from, uint amount);  
    event transferred(address from, address to, uint amount);  
  
    constructor(uint _total) public {  
        balances[msg.sender] = total = _total;  
        emit initialized(msg.sender, total);  
    }  
  
    function transfer(address to, uint amount) public {  
        require(balances[msg.sender] >= amount && msg.sender != to);  
        balances[msg.sender] -= amount;  
        balances[to] += amount;  
        emit transferred(msg.sender, to, amount);  
    }  
}
```

Solidity Events

- Stored in blockchain logs
- Contract communicates with user
 - Important state changes
- Abstract view of execution
 - Relevant aspect to each user



Motivation

Do we always emit if balances change?

Was there a change when emitted?

Can we trust (rely on)
the emitted events?

Is the amount correct?

Not really...

Formal Specification of Events

- What state variable(s) do events **track**?
 - Emit event *iff* there was a change

```
contract Token {
  mapping(address=>uint) balances;
  uint total;

  /// @notice tracks-changes-in balances
  /// @notice tracks-changes-in total
  event initialized(address from, uint amount);

  /// @notice tracks-changes-in balances
  event transferred(address from, address to, uint amount);
}
```

Formal Specification of Events

- What events *can* functions emit?
 - Similar to Java *throws*

```
contract Token {  
    /// @notice emits initialized  
    constructor(uint _total) public {  
        ...  
    }  
  
    /// @notice emits transferred  
    function transfer(address to, uint amount) public {  
        ...  
    }  
}
```

Formal Specification of Events

- What are the **conditions** *before* and *at* the emit?

```
contract Token {  
    /// @notice precondition balances[from] == 0  
    /// @notice postcondition balances[from] == amount  
    /// @notice postcondition total == amount  
    event initialized(address from, uint amount);  
  
    /// @notice precondition balances[from] >= amount  
    /// @notice postcondition balances[from] == before(balances[from]) - amount  
    /// @notice postcondition balances[to] == before(balances[to]) + amount  
    event transferred(address from, address to, uint amount);  
}
```




DEMO

Formal Verification

- Where to check if an event has been **emitted**?
 - Cannot check immediately (modification in multiple steps)
- Where to check **preconditions**?
 - What does “before the change” exactly mean?



Checkpoints

```
function transfer(address to, uint amount) public {  
    require(balances[msg.sender] >= amount && msg.sender != to);  
    ...  
    balances[msg.sender] -= amount;  
    balances[to] += amount;  
    ...  
    emit transferred(msg.sender, to, amount);  
    ...  
}
```

Before checkpoint

- First time variable changes
- Save state (for precondition)

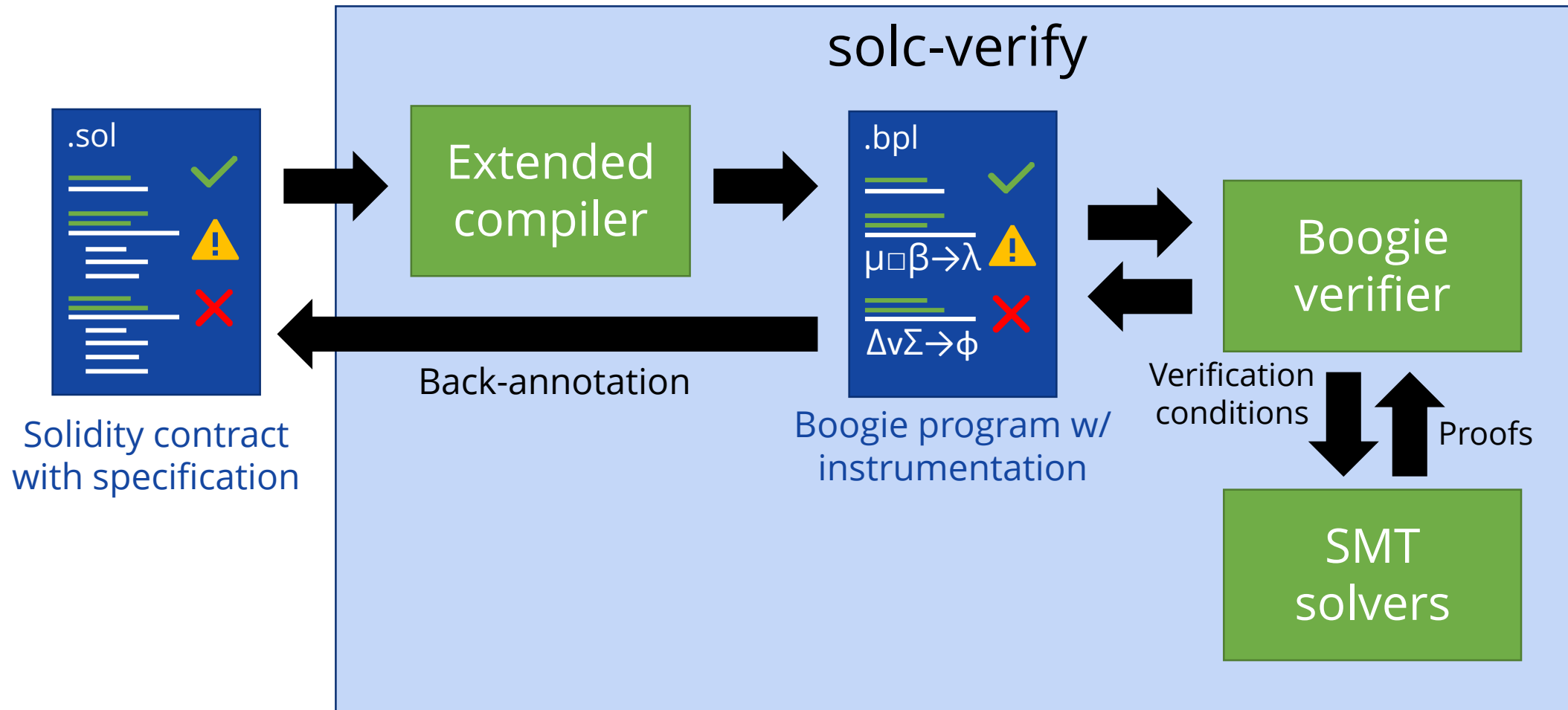
Emit

- Check pre/post
- Clear before/after checkpoint

After checkpoint

- Static barrier
- Latest point to emit
- E.g., function end

Overview



Instrumentation

```
mapping(address=>uint) balances;
```

new vars

```
/// @notice emits transferred
```

```
function transfer(address to, uint amount) public {
```

assume clear

```
require(balances[msg.sender] >= amount  
&& msg.sender != to);
```

check modif

```
balances[msg.sender] -= amount;
```

check modif

```
balances[to] += amount;
```

emit specs

```
emit transferred(msg.sender, to, amount);
```

after checkpt

```
}
```

```
assert(!bal_modif);
```

```
mapping(address=>uint) bal_old;  
bool bal_modif;
```

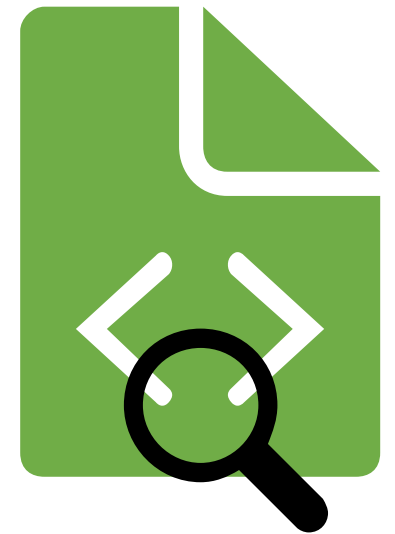
```
require(!bal_modif);
```

```
if (!bal_modif) {  
    bal_old = balances;  
    bal_modif = true; }
```

```
assert(bal_modif);  
assert(bal_old[msg.sender] >= amount);  
assert(balances[msg.sender] == bal_old[msg.sender]-amount);  
assert(balances[to] == bal_old[to] + amount);  
bal_modif = false;
```

Discussion

- We used solc-verify
 - Modular verifier based on Boogie and SMT
 - Can work with [other verifiers](#) (supporting assertions)
- After checkpoints
 - Depend on [verification approach](#)
 - Modular verification: loop boundaries as well

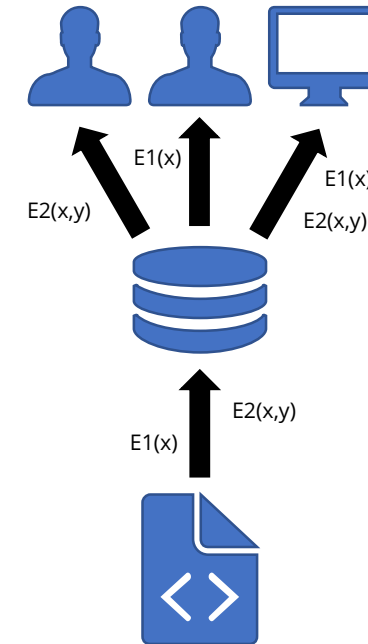


Conclusions

- Solidity **events** provide abstract view
- Formal **specification** and **verification**
- In-code **annotations**
- Checkpoints
- Instrumentation

arxiv.org/abs/2005.10382

github.com/SRI-CSL/solidity



```
contract Token {
  mapping(address=>uint) balances;
  uint total;

  /// @notice tracks-changes-in balances
  /// @notice tracks-changes-in total
  event initialized(address from, uint amount);

  /// @notice tracks-changes-in balances
  event transferred(address from, address to, uint amount);
}
```