solc-verify: source-level formal verification for Solidity

Ákos Hajdu[1], Dejan Jovanović[2]

[1] *Budapest University of Technology and Economics*
[2] *SRI International*

MŰEGYETEM 1782

SRI International

Nemzeti Kutatási, Fejlesztési és Innovációs Hivatal

ÚNKP Új Nemzeti Kiválósági Program

Partially supported by the ÚNKP-19-3 New National Excellence Program of the Ministry for Innovation and Technology.

Verification Landscape

KSolidity

SMTChecker

KEVM

VeriSolid

Slither

solc-verify

MythX

Certora

VerX

Securify

VeriSol

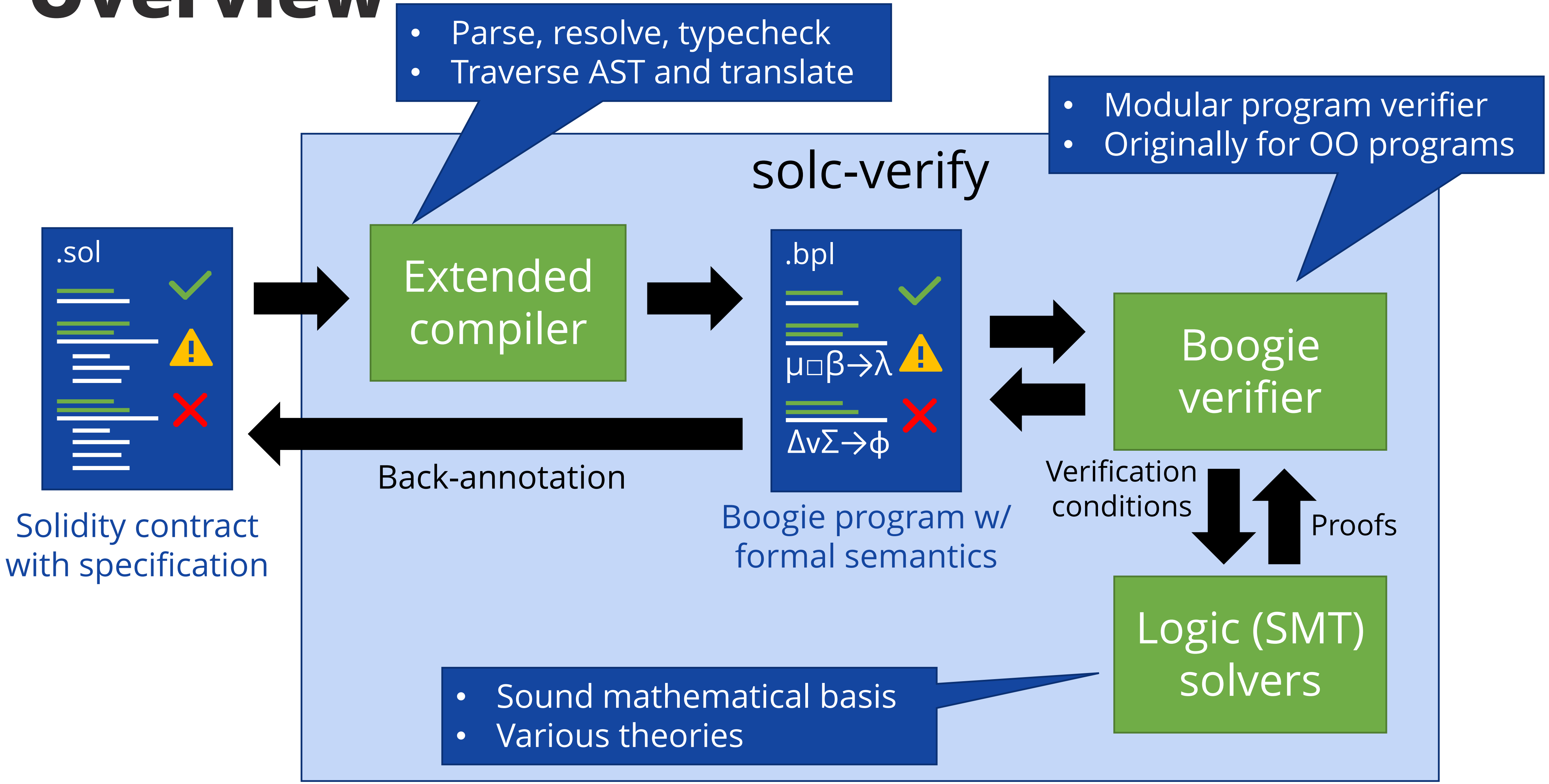
Truffle

… and many more

DEMO

github.com/hajduakos/solidity-summit-demo

Overview



- Parse, resolve, typecheck
- Traverse AST and translate

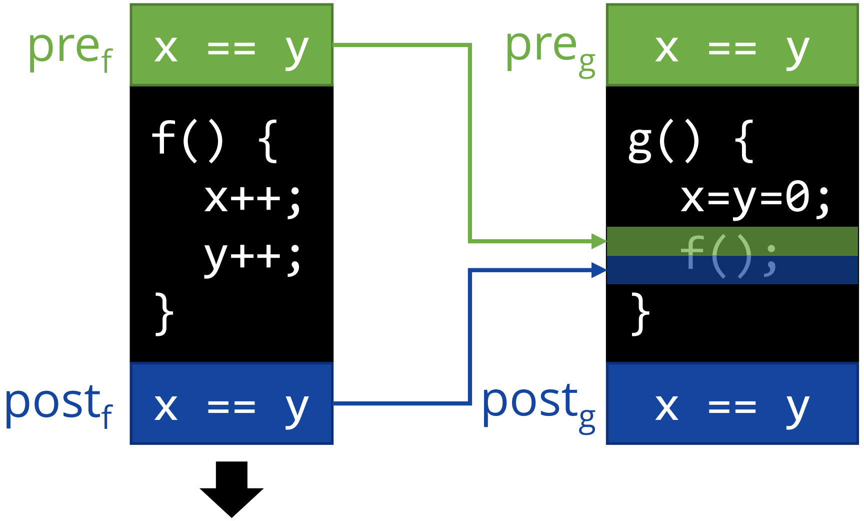
- Modular program verifier
- Originally for OO programs

- Sound mathematical basis
- Various theories

Formal Verification

- Functional correctness w.r.t specification
  - Implicit: assertion, overflow
  - Explicit: pre/postconditions, invariants, ...

- Modular verification
  - pre + body → post
  - Discharge with SMT solvers
  - Replace calls with specification



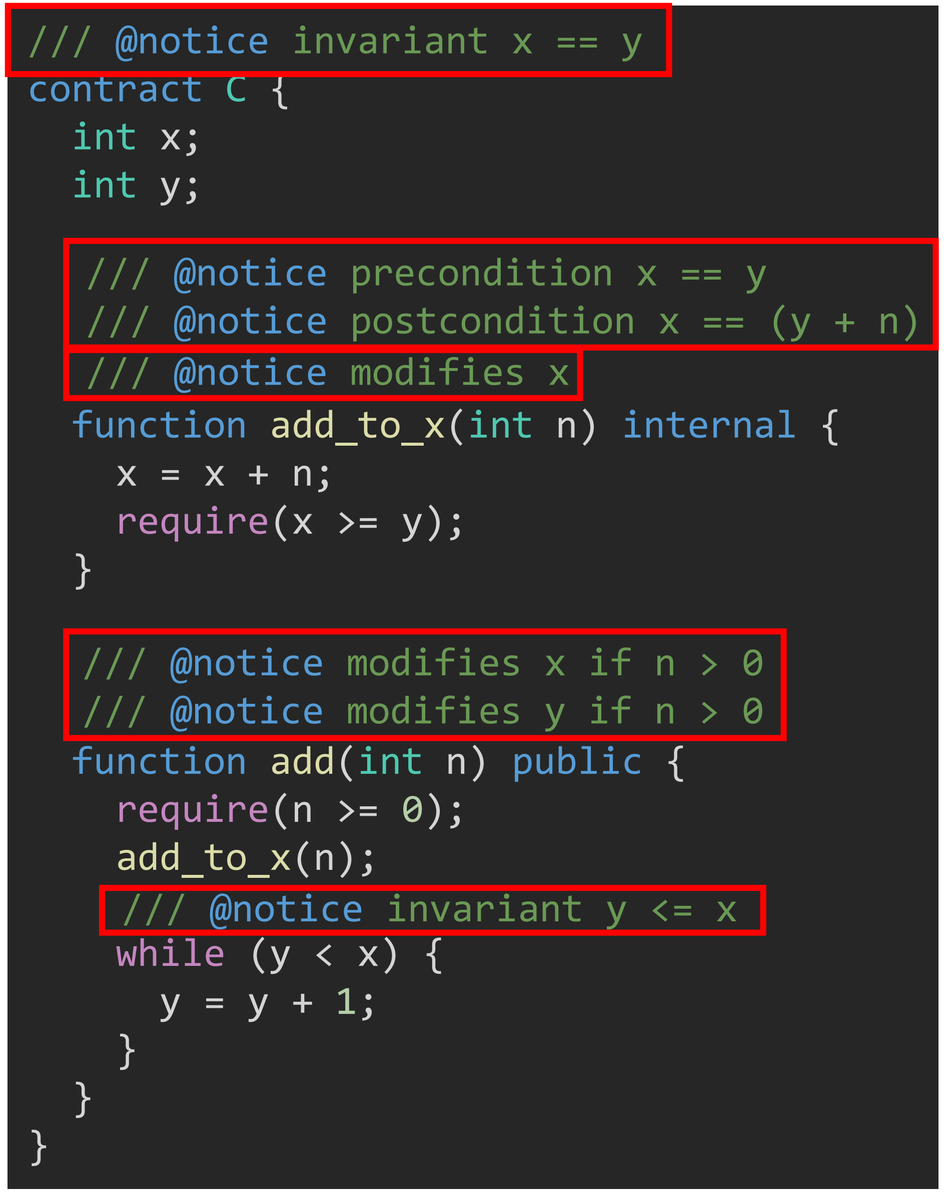
```
f() {
   x++;
   y++;
}
```

```
g() {
   x=y=0;
   f();
}
```

If $x_0 = y_0$ and $x_1 = x_0 + 1$ and $y_1 = y_0 + 1$
then $x_1 = y_1$?

Specification Annotations

- Solidity provides (implicit)
  - require, assert
- Annotation language (explicit)
  - Features
    - Pre/postconditions
    - Contract level invariants
    - Loop invariants
    - Access control (modifies)
    - Events *in progress*
  - Solidity expressions (side effect free)
    - Extra: sum over collections, previous values
    - Quantifiers *in progress*



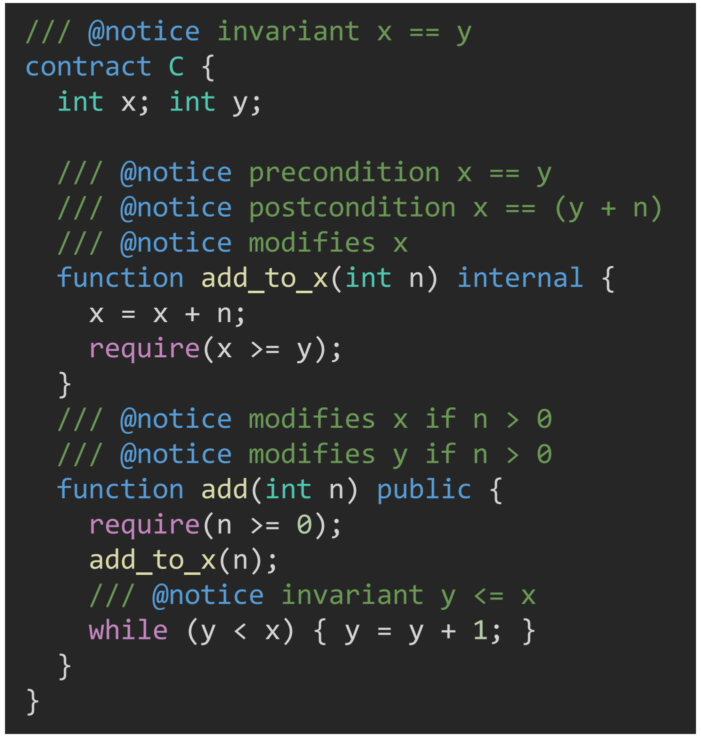
```
/// @notice invariant x == y
contract C {
  int x;
  int y;

  /// @notice precondition x == y
  /// @notice postcondition x == (y + n)
  /// @notice modifies x
  function add_to_x(int n) internal {
    x = x + n;
    require(x >= y);
  }

  /// @notice modifies x if n > 0
  /// @notice modifies y if n > 0
  function add(int n) public {
    require(n >= 0);
    add_to_x(n);
    /// @notice invariant y <= x
    while (y < x) {
      y = y + 1;
    }
  }
}
```

DEMO

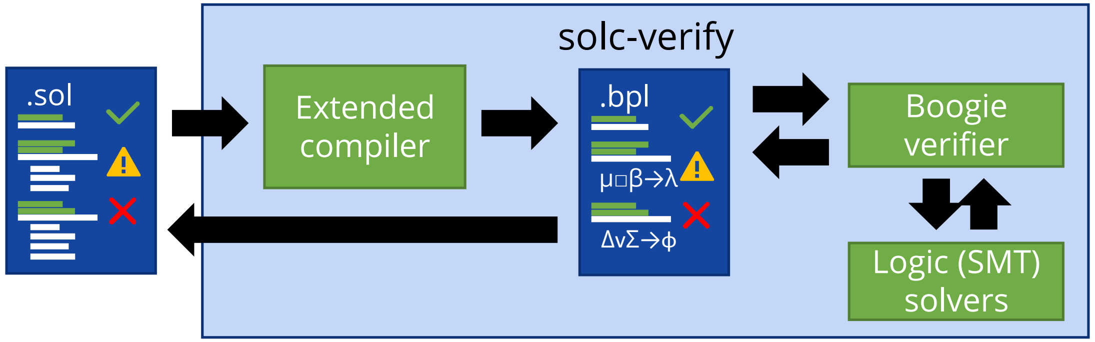
github.com/hajduakos/solidity-summit-demo

Summary

- solc-verify: source-level formal verification for Solidity



```
/// @notice invariant x == y
contract C {
  int x; int y;

  /// @notice precondition x == y
  /// @notice postcondition x == (y + n)
  /// @notice modifies x
  function add_to_x(int n) internal {
    x = x + n;
    require(x >= y);
  }
  /// @notice modifies x if n > 0
  /// @notice modifies y if n > 0
  function add(int n) public {
    require(n >= 0);
    add_to_x(n);
    /// @notice invariant y <= x
    while (y < x) { y = y + 1; }
  }
}
```



github.com/SRI-CSL/solidity

hajduakos.github.io

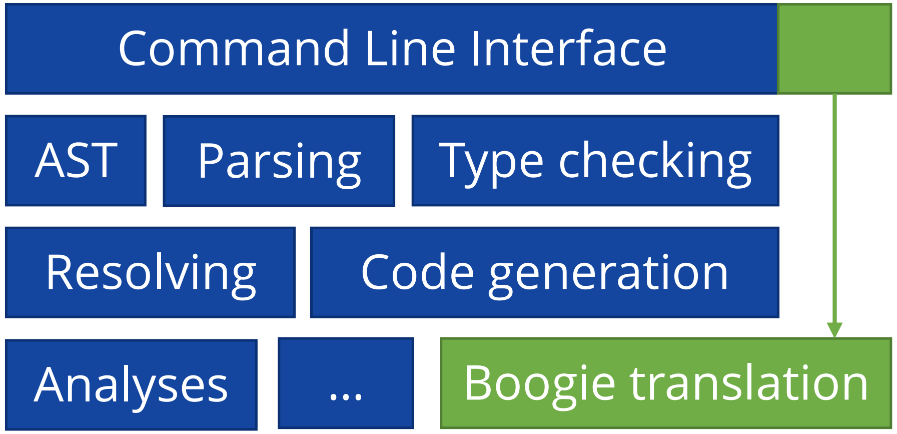
@himynameisakos

Examples: github.com/hajduakos/solidity-summit-demo
Tool paper: arxiv.org/abs/1907.04262
Formalizing memory model: arxiv.org/abs/2001.03256

Without Modifying the Compiler?

- More like extending than modifying
  - Compiler works same, with new options
- In principle could be done externally
  - Parse JSON AST
- Benefits of being inside compiler
  - More robust to changes, type safety
  - Reuse modules (e.g. parse specs)

- Extensible compiler infrastructure?
  - Like LLVM



github.com/ethereum/act

Relationship with Act

- Act
  - Language independent
  - Separate specs
- solc-verify
  - Specs in the same language, Solidity
  - Code and specs together
- In principle, specs could come from (a subset) of Act

```
behaviour init of StateMachine
interface constructor()

creates
        uint256 x := 0
invariants
        x <= 1

behaviour f of StateMachine
interface f()

case x == 0:
        storage
                x => 1
case _:
        noop
ensures
        (x == 0) or (x == 1)
```

Relationship with SMTChecker

- SMTChecker (to the best of our knowledge)
  - Built-in analyzer in the compiler
  - Direct translation to SMT
  - Implicit specifications (overflow, assert)
  - Intra-function analysis
- We did some experiments
  - v0.5.10: arxiv.org/abs/1907.04262
  - v0.5.12: arxiv.org/abs/2001.03256
  - Unsupported features, false alarms for our use cases
    - E.g., overflows, memory model, external calls