Program Testing and Analysis:
Symbolic and Concolic Testing

Prof. Dr. Michael Pradel
Software Lab, TU Darmstadt
Outline

1. Classical **Symbolic Execution**
2. **Challenges** of Symbolic Execution
3. **Concolic** Testing
4. Large-Scale Application in **Practice**

Mostly based on these papers:

- *DART: directed automated random testing*, Godefroid et al., PLDI’05
- *KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs*, Cadar et al., OSDI’08
- *Automated Whitebox Fuzz Testing*, Godefroid et al., NDSS’08
Symbolic Execution

- Reason about behavior of program by "executing" it with symbolic values
- Originally proposed by James King (1976, CACM) and Lori Clarke (1976, IEEE TSE)
- Became practical around 2005 because of advances in constraint solving (SMT solvers)
function f(a, b, c) {
    var x = y = z = 0;
    if (a) {
        x = -2;
    }
    if (b > 5) {
        if (!a && c) {
            y = 1;
        }
        z = 2;
    }
    assert(x + y + z != 3);
}
function f(a, b, c) {
    var x = y = z = 0;
    if (a) {
        x = -2;
    }
    if (b < 5) {
        if (!a && c) {
            y = 1;
        }
        z = 2;
    }
    assert(x + y + z != 3);
}

concrete execution

\[
\begin{align*}
    a &= b = c = 1 \\
    x &= y = z = 0 \\
    \text{true} \\
    x &= -2 \\
    \text{true} \\
    \text{false} \\
    z &= 2 \\
    -2 + 0 + 2 &\neq 3 \quad \checkmark
\end{align*}
\]
Execution Tree

All possible execution paths

- Binary tree
- Nodes: Conditional statements
- Edges: Execution of sequence on non-conditional statements
- Each path in the tree represents an equivalence class of inputs
function f(a, b, c) {
    var x = y = z = 0;
    if (a) {
        x = -2;
    }
    if (b < 5) {
        if (!a && c) {
            y = 1;
        }
        z = 2;
    }
    assert(x + y + z != 3); y = 1; z = 2;
}
Quiz

Draw the execution tree for this function. How many nodes and edges does it have?

```javascript
function f(x, y) {
  var s = "foo";
  if (x < y) {
    s += "bar";
    console.log(s);
  }
  if (y === 23) {
    console.log(s);
  }
}
```
function f(x, y) {
    var s = "foo";
    if (x < y) {
        s += "bar";
        console.log(s);
    }
    if (y === 23) {
        console.log(s);
    }
}

\[
x = x_0 \\
y = y_0 \\
s = \text{"foo"} \\
x_0 < y_0 \\
\]

\[s += \text{"bar"} \\
t \\\n\text{cons...} \\
y_0 = 23 \\
\]

\[t \\\nf \\\n\text{cons...} \\
y_0 = 23 \\
\]

\[t \\\nf \\\n\text{cons...} \\
f \\\n\]

\[\rightarrow 3 \text{ nodes, 7 edges} \]
Symbolic Values and Symbolic State

- Unknown values, e.g., user inputs, are kept symbolically
- Symbolic state maps variables to symbolic values

```javascript
function f(x, y) {
  var z = x + y;
  if (z > 0) {
    ...
  }
}
```
Symbolic Values and Symbolic State

- **Unknown values**, e.g., user inputs, are kept symbolically
- **Symbolic state** maps variables to symbolic values

```javascript
function f(x, y) {
    var z = x + y;
    if (z > 0) {
        ...
    }
}
```

**Symbolic input values**: $x_0$, $y_0$

**Symbolic state**: $z = x_0 + y_0$
Path Conditions

Quantifier-free formula over the symbolic inputs that encodes all branch decisions taken so far

```javascript
function f(x, y) {
    var z = x + y;
    if (z > 0) {
        ...
    }
}
```
Path Conditions

Quantifier-free formula over the symbolic inputs that encodes all branch decisions taken so far

```javascript
function f(x, y) {
  var z = x + y;
  if (z > 0) {
    ...
  }
}
```

Path condition:

\[ x_0 + y_0 > 0 \]
Satisfiability of Formulas

Determine whether a path is **feasible**: Check if its path condition is satisfiable

- Done by powerful **SMT/SAT solvers**
  - SAT = satisfiability,
  - SMT = satisfiability modulo theory
  - E.g., Z3, Yices, STP
- For a satisfiable formula, solvers also provide a **concrete solution**
- **Examples:**
  - \( a_0 + b_0 > 1 \): Satisfiable, one solution: \( a_0 = 1, b_0 = 1 \)
  - \((a_0 + b_0 < 0) \land (a_0 - 1 > 5) \land (b_0 > 0)\): Unsatisfiable