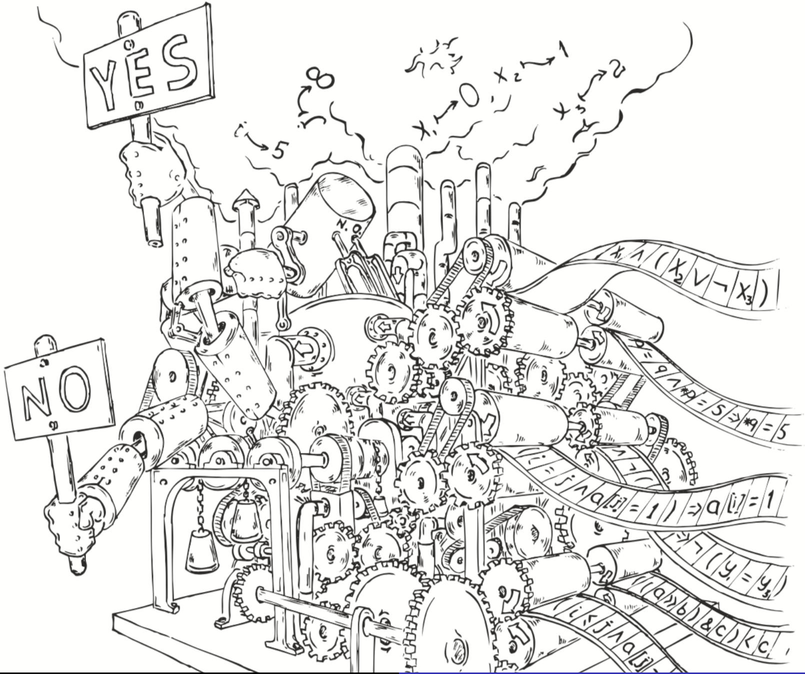

YES
NO

SAT/SMT solvers
11. Symbolic execution

Roman Kholin

Lomonosov Moscow State University

Moscow, 2023

Whe could used

- Dynamic analysis
- Program correctness
- Test generations
- Taint analysis

Concrete execution

```
void foo(int x, int y) {
    int t = 0;
    if (x > y) {
        t = x;
    } else {
        t = y;
    }
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
4	4	0

```
void foo(int x, int y) {
    int t = 0;
    //if (x > y) {
    //    t = x;
    //} else {
        t = y;
    //}
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
4	4	4

Concrete execution

```
void foo(int x, int y) {
    int t = 0;
    if (x > y) {
        t = x;
    } else {
        t = y;
    }
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
2	1	0

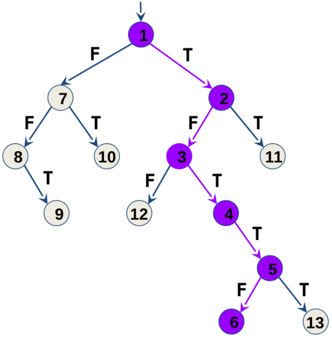
Concrete execution

```
void foo(int x, int y) {
    int t = 0;
    //if (x > y) {
        t = x;
    //} else {
    //    t = y;
    //}
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
2	1	2

Execution pathes

- Program could be representerd as binary tree - computed tree
- Every node corespond to condition operator
- Every edge corespond to execution of comands (each of then is not a condition operator)
- Each path from root to leaf correspond to equivalence class



Example

```
void test(int x, int y) {
    if (2*y == x) {
        if (x <= y+10) {
            printf("OK");
        } else {
            printf("not OK");
            assert false;
        }
    } else {
        print("OK");
    }
}
```

Existed approach

```
void test(int x) {
    if (x == 94389) {
        assert false;
    }
}
```

- Random testion
- The probability to detect error is very low

Symbolic execution

```
void foo(int x, int y) {
    int t = 0;
    if (x > y) {
        t = x;
    } else {
        t = y;
    }
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
x	y	0

Symbolic execution

```
void foo(int x, int y) {
    int t = 0;
    if (x > y) {
        t = x;
    } else {
        t = y;
    }
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
x	y	t_0

$$t_0 = \begin{cases} \mathrm{x} & x > y \\ \mathrm{y} & x \leq y \end{cases}$$

Symbolic execution

```
void foo(int x, int y) {
    int t = 0;
    if (x > y) {
        t = x;
    } else {
        t = y;
    }
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
x	y	t_0

$$t_0 = \begin{cases} \mathrm{x} & x > y \\ \mathrm{y} & x \leq y \end{cases}$$

$$t_0 < x?$$

```
void foo(int x, int y) {
    int t = 0;
    if (x > y) {
        t = x;
    } else {
        t = y;
    }
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
x	y	t_0

$$t_0 = \begin{cases} \mathrm{x} & x > y \\ \mathrm{y} & x \leq y \end{cases}$$

$$\begin{cases} x > y \implies t_0 = x \implies t_0 \geq x \\ x \leq y \implies t_0 = y \implies t_0 \geq x \end{cases}$$

```
void foo(int x, int y) {
    int t = 0;
    if (x > y) {
        t = x - 1;
    } else {
        t = y;
    }
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
x	y	t_0

Symbolic execution

```c
void foo(int x, int y) {
    int t = 0;
    if (x > y) {
        t = x - 1;
    } else {
        t = y;
    }
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
x	y	t_0

$$t_0 = \begin{cases} \text{x - 1} & x > y \\ \text{y} & x \leq y \end{cases}$$

```
void foo(int x, int y) {
    int t = 0;
    //if (x > y) {
        t = x - 1;
    //} else {
    //    t = y;
    //}
    //if (t < x) {
        assert false;
    //}
}
```

X	Y	T
x	y	t_0

$$t_0 = \begin{cases} \text{x - 1} & x > y \\ \text{y} & x \leq y \end{cases}$$

$$\begin{cases} x > y \implies t_0 = x - 1 \implies t_0 < x \\ x \leq y \implies t_0 = y \implies t_0 \geq x \end{cases}$$

$x > y$ - solution

Symbolic execution

```
void testme(int x) {
    if (pow(2,x) % c == 17) {
        printf("not OK");
        assert false;
    } else
        printf("OK");
}
```

Concolic execution

Concolic execution (or dynamic symbolic execution):

- Start with random input data
- Track the conrete and symbolic variables

Concolic execution

```
void foo(int x, int y) {
    int t = 0;
    if (x > y) {
        t = x;
    } else {
        t = y;
    }
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
x	y	t_0

$$t_0 = \begin{cases} \mathrm{x} & x > y \\ \mathrm{y} & x \leq y \end{cases}$$

Concolic execution

```
void foo(int x, int y) {
    int t = 0;
    if (x > y) {
        t = x;
    } else {
        t = y;
    }
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
$(0, x)$	$(0, y)$	$(0, 0)$

Concolic execution

```
void foo(int x, int y) {
    int t = 0;
    if (x > y) {
        t = x;
    } else {
        t = y;
    }
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
$(0, x)$	$(0, y)$	$(0, 0)$

Concolic execution

```
void foo(int x, int y) {
    int t = 0;
    if (x > y) {
        t = x;
    } else {
        t = y;
    }
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
$(0, x)$	$(0, y)$	$(0, 0)$

$\{\ F1 = not(x > y)$

Concolic execution

```
void foo(int x, int y) {
    int t = 0;
    if (x > y) {
        t = x;
    } else {
        t = y;
    }
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
$(0, x)$	$(0, y)$	$(0, 0)$

$\{\ F1 = not(x > y)$

$SMT\_Solver(\text{not}\ F1) \rightarrow (x = 1, y = 0)$

Concolic execution

```
void foo(int x, int y) {
    int t = 0;
    if (x > y) {
        t = x;
    } else {
        t = y;
    }
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
$(0, x)$	$(0, y)$	$(0, 0)$

$\{\ F1 = not(x > y)$

$SMT\_Solver(\text{not}$

$F1) \rightarrow (x = 1, y = 0)$

$queue = \{(x = 1, y = 0)\}$

Concolic execution

```
void foo(int x, int y) {
    int t = 0;
    if (x > y) {
        t = x;
    } else {
        t = y;
    }
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
$(0, x)$	$(0, y)$	$(0, 0)$

$$\left\{ \; F1 = not(x > y) \right.$$

$$queue = \{(x = 1, y = 0)\}$$

Concolic execution

```
void foo(int x, int y) {
    int t = 0;
    if (x > y) {
        t = x;
    } else {
        t = y;
    }
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
$(0, x)$	$(0, y)$	$(0, y)$

$$\left\{ \ F1 = not(x > y) \right.$$

$$queue = \{(x = 1, y = 0)\}$$

Concolic execution

```
void foo(int x, int y) {
    int t = 0;
    if (x > y) {
        t = x;
    } else {
        t = y;
    }
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
$(0, x)$	$(0, y)$	$(0, y)$

$$\{\ F1 = not(x > y)$$

$$queue = \{(x = 1, y = 0)\}$$

Concolic execution

```
void foo(int x, int y) {
    int t = 0;
    if (x > y) {
        t = x;
    } else {
        t = y;
    }
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
$(0, x)$	$(0, y)$	$(0, y)$

$$\{\ F1 = not(x > y)$$

$$queue = \{(x = 1, y = 0)\}$$

Concolic execution

```
void foo(int x, int y) {
    int t = 0;
    if (x > y) {
        t = x;
    } else {
        t = y;
    }
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
$(0, x)$	$(0, y)$	$(0, y)$

$$\begin{cases} F1 = not(x > y) \\ F2 = not(y < x) \end{cases}$$

$$queue = \{(x = 1, y = 0)\}$$

Concolic execution

```
void foo(int x, int y) {
    int t = 0;
    if (x > y) {
        t = x;
    } else {
        t = y;
    }
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
$(0, x)$	$(0, y)$	$(0, y)$

$$\begin{cases} F1 = not(x > y) \\ F2 = not(y < x) \end{cases}$$

$SMT\_Solver(F1$ and not $F2) \rightarrow UNSAT$

$queue = \{(x = 1, y = 0)\}$

Concolic execution

```
void foo(int x, int y) {
    int t = 0;
    if (x > y) {
        t = x;
    } else {
        t = y;
    }
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
$(0, x)$	$(0, y)$	$(0, y)$

$$\begin{cases} F1 = not(x > y) \\ F2 = not(y < x) \end{cases}$$

$$queue = \{(x = 1, y = 0)\}$$

Concolic execution

```
void foo(int x, int y) {
    int t = 0;
    if (x > y) {
        t = x;
    } else {
        t = y;
    }
    if (t < x) {
        assert false;
    }
}
```

X	Y	T
$(1, x)$	$(0, y)$	$(0, 0)$

$queue = \{\}$

Example

```
int test(int x) {
    int[] A = { 5, 7, 9 };
    int i = 0;
    while (i < 3) {
        if (A[i] == x) {
            break;
        }
        i++;
    }
    return i;
}
```

```
int foo(int v) {
    return secure_hash(v);
}

void test(int x, int y) {
    if (x != y)
        if (foo(x) == foo(y))
            assert;
}
```

Consolic Execution

- Could never stop execution
- Sound
- Not complete

Implementation

```
a = b + c
```

Implementation

```
class concolic_int(int):
def __new__(cls, val, sym):
    self =
     super(concolic_int, cls).__new__(cls, val)
    self.__val = val
    self.__sym = sym
    return self
def __add__(self, other):
    if isinstance(other, concolic_int):
        value = self.__val + other.__val
        symbolic = self.__sym + "+" + other.__sym
    else:
        value = self.__val + other
        symbolic = self.__sym + "+" + str(other)
    return concolic_int(value, symbolic)
```

Implementation

How could one change int on concolic_int?

Implementation

```
a = b + c
```

Implementation

```
a = plus(b, c)
```

Implementation

```
function plus(x, y) {
    if (x isinstanceof Concolic) {
        if (y isinstanceof Concolic) {
            return new Concolic(
                x._val + y._val,
                x._sym + "+" + y._sym
            );
        } else {
            return new Concolic(
                x._val + y,
                x._sym + "+" + y.toString()
            );
        }
    } else {
        ....
    }
}
```

Implementation

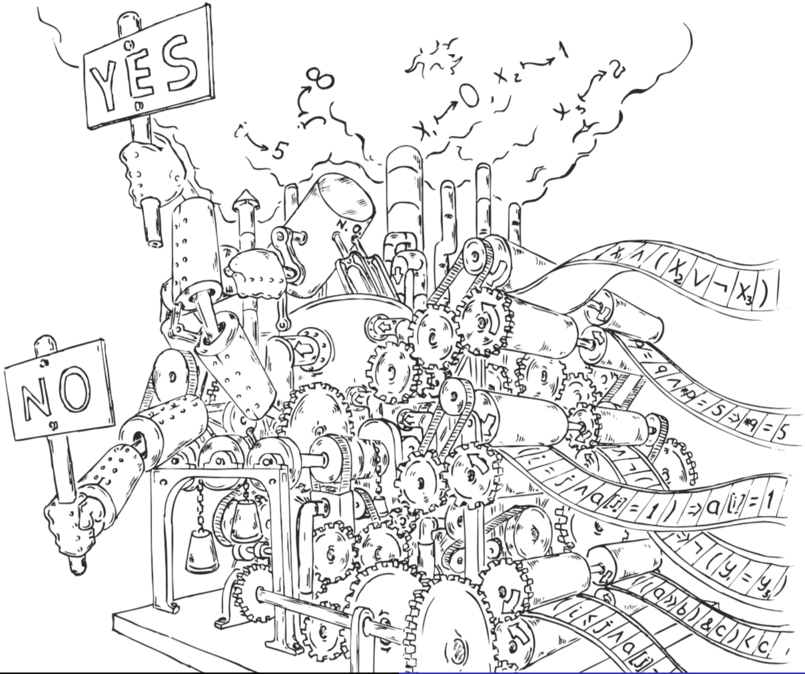
How to ecode the pathes?

Examples

- KLEE: LLVM (C family of languages)
- PEX: .NET Framework
- CUTE: C
- jCUTE: Java
- Jalangi: Javascript
- Jalangi2 + ExpoSE: Javascript
- SAGE and S2E: binaries (x86, ARM, ...)

Links

- https://www.youtube.com/watch?v=yRVZPvHYHzw - MIT lecture
- Symbolic Execution and Program Testing. James C. King
- SAGE: Whitebox Fuzzing for Security Testing. Patrice Godefroid, Michael Y. Levin, and David A. Molnar
- Jalangi: A Selective Record-Replay and Dynamic Analysis Framework for JavaScript. Koushik Sen, Swaroop Kalasapur, Tasneem Brutch, Simon Gibbs
- Sound Regular Expression Semantics for Dynamic Symbolic Execution of JavaScript. Blake Loring, Duncan Mitchell, Johannes Kinder


YES
NO
N.O.
i → 5
x₁ → 0
x₂ → 1
x₃ → 2
∧(x₂∨¬x₃)
⇒a[i]=1
¬(y₁=y₃)
((a>>b)&c)<c