



Задачи разрешимости логических формул и приложения

Лекция 11. Символическое исполнение

Роман Холин

Московский государственный университет

Москва, 2021

- Хочется автоматизировать генерацию тестовых примеров

- Хочется автоматизировать генерацию тестовых примеров
- Хочется, чтобы тесты посещали все строки кода (LOC метрика)

- Хочется автоматизировать генерацию тестовых примеров
- Хочется, чтобы тесты посещали все строчки кода (LOC метрика)
- Хочется, чтобы тесты посещали все пути обхода программы

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

Конкретное исполнение (Concrete execution)

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     if (x > y) {  
4         t = x;  
5     } else {  
6         t = y;  
7     }  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

X	Y	T
4	4	0

Конкретное исполнение (Concrete execution)

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     //if (x > y) {  
4         //     t = x;  
5     //} else {  
6         t = y;  
7     //}  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

X	Y	T
4	4	4

Конкретное исполнение (Concrete execution)

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     if (x > y) {  
4         t = x;  
5     } else {  
6         t = y;  
7     }  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

X	Y	T
2	1	0

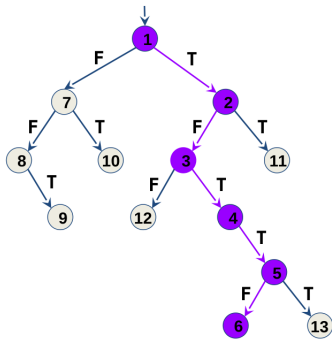
Конкретное исполнение (Concrete execution)

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     //if (x > y) {  
4         t = x;  
5     //} else {  
6         //     t = y;  
7     //}  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

X	Y	T
2	1	2

Пути исполнения программы

- Программа может быть представлена в виде бинарного дерева - т.н. Вычислительного дерева
- Каждая вершина - выполнение условного оператора
- Каждое ребро - выполнение последовательности команд, которые не являются условным оператором
- Каждый путь от корня - делит множество входных данных на классы эквивалентности



Пример дерева

```
1 void test(int x, int y) {
2     if (2*y == x) {
3         if (x <= y+10) {
4             printf("OK");
5         } else {
6             printf("not OK");
7             assert false;
8         }
9     } else {
10        print("OK");
11    }
12 }
```

```
1 void test(int x) {  
2     if (x == 94389) {  
3         assert false;  
4     }  
5 }
```

- Рандомизированное тестирование
- Проблема: вероятность ошибка очень мала

Символическое исполнение (Symbolic execution)

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     if (x > y) {  
4         t = x;  
5     } else {  
6         t = y;  
7     }  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

X	Y	T
x	y	0

Символическое исполнение (Symbolic execution)

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     if (x > y) {  
4         t = x;  
5     } else {  
6         t = y;  
7     }  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

X	Y	T
x	y	t_0

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

Символическое исполнение (Symbolic execution)

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     if (x > y) {  
4         t = x;  
5     } else {  
6         t = y;  
7     }  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

X	Y	T
x	y	t_0

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

$$t_0 < x?$$

Символическое исполнение (Symbolic execution)

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     if (x > y) {  
4         t = x;  
5     } else {  
6         t = y;  
7     }  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

X	Y	T
x	y	t_0

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

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

Символическое исполнение (Symbolic execution)

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     if (x > y) {  
4         t = x - 1;  
5     } else {  
6         t = y;  
7     }  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

X	Y	T
x	y	t_0

Символическое исполнение (Symbolic execution)

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     if (x > y) {  
4         t = x - 1;  
5     } else {  
6         t = y;  
7     }  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

X	Y	T
x	y	t_0

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

Символическое исполнение (Symbolic execution)

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     //if (x > y) {  
4         t = x - 1;  
5     //} else {  
6         //     t = y;  
7     //}  
8     //if (t < x) {  
9         assert false;  
10    //}  
11 }
```

X	Y	T
x	y	t_0

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

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

$x > y$ - solution

Символическое исполнение (Symbolic execution)

```
1 void testme(int x) {  
2     if (pow(2,x) % c == 17) {  
3         printf("not OK");  
4         assert false;  
5     } else  
6         printf("OK");  
7 }
```

Concolic execution (или dynamic symbolic execution):

- Начнем с случайных входных данных
- Поддерживаем конкретные и символические переменные

Concolic execution

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     if (x > y) {  
4         t = x;  
5     } else {  
6         t = y;  
7     }  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

X	Y	T
x	y	t_0

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

Concolic execution

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     if (x > y) {  
4         t = x;  
5     } else {  
6         t = y;  
7     }  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

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

Concolic execution

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     if (x > y) {  
4         t = x;  
5     } else {  
6         t = y;  
7     }  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

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

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     if (x > y) {  
4         t = x;  
5     } else {  
6         t = y;  
7     }  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

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

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

Concolic execution

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     if (x > y) {  
4         t = x;  
5     } else {  
6         t = y;  
7     }  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

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

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

Concolic execution

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     if (x > y) {  
4         t = x;  
5     } else {  
6         t = y;  
7     }  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

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

$\{ F1 = \text{not}(x > y)$
 $\text{SMT_Solver}(\text{not}$
 $F1) \rightarrow (x = 1, y = 0)$
 $\text{queue} = \{(x = 1, y = 0)\}$

Concolic execution

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     if (x > y) {  
4         t = x;  
5     } else {  
6         t = y;  
7     }  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

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

$\{ F1 = \text{not}(x > y)$
 $\text{queue} = \{(x = 1, y = 0)\}$

Concolic execution

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     if (x > y) {  
4         t = x;  
5     } else {  
6         t = y;  
7     }  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

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

$\{ F1 = \text{not}(x > y)$
 $\text{queue} = \{(x = 1, y = 0)\}$

Concolic execution

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     if (x > y) {  
4         t = x;  
5     } else {  
6         t = y;  
7     }  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

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

$\{ F1 = \text{not}(x > y)$
 $\text{queue} = \{(x = 1, y = 0)\}$

Concolic execution

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     if (x > y) {  
4         t = x;  
5     } else {  
6         t = y;  
7     }  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

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

$\{ F1 = \text{not}(x > y)$
 $\text{queue} = \{(x = 1, y = 0)\}$

Concolic execution

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     if (x > y) {  
4         t = x;  
5     } else {  
6         t = y;  
7     }  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

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

$\begin{cases} F1 = \text{not}(x > y) \\ F2 = \text{not}(y < x) \end{cases}$
 $queue = \{(x = 1, y = 0)\}$

Concolic execution

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     if (x > y) {  
4         t = x;  
5     } else {  
6         t = y;  
7     }  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

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

$\begin{cases} F1 = \text{not}(x > y) \\ F2 = \text{not}(y < x) \end{cases}$
 $SMT_Solver(F1 \text{ and not } F2) \rightarrow UNSAT$
 $queue = \{(x = 1, y = 0)\}$

Concolic execution

```
1 void foo(int x, int y) {  
2     int t = 0;  
3     if (x > y) {  
4         t = x;  
5     } else {  
6         t = y;  
7     }  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

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

$\begin{cases} F1 = \text{not}(x > y) \\ F2 = \text{not}(y < x) \end{cases}$
 $queue = \{(x = 1, y = 0)\}$

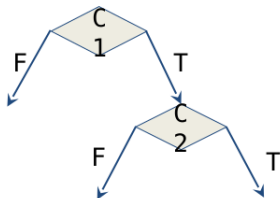
```
1 void foo(int x, int y) {  
2     int t = 0;  
3     if (x > y) {  
4         t = x;  
5     } else {  
6         t = y;  
7     }  
8     if (t < x) {  
9         assert false;  
10    }  
11 }
```

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

$queue = \{\}$

Какие ограничения могут повстречаться при consolic execution?

- | | |
|------------------------------------|---|
| <input type="checkbox"/> C1 | <input type="checkbox"/> $C1 \wedge C2$ |
| <input type="checkbox"/> C2 | <input type="checkbox"/> $C1 \wedge \neg C2$ |
| <input type="checkbox"/> $\neg C1$ | <input type="checkbox"/> $\neg C1 \wedge C2$ |
| <input type="checkbox"/> $\neg C2$ | <input type="checkbox"/> $\neg C1 \wedge \neg C2$ |



Пример

```
1  int test(int x) {
2      int [] A = { 5, 7, 9 };
3      int i = 0;
4      while (i < 3) {
5          if (A[i] == x) {
6              break;
7          }
8          i++;
9      }
10     return i;
11 }
```

Какие ограничения появлялись
и были разрешены солвером?

Пример

```
1 int foo(int v) {  
2     return secure_hash(v);  
3 }  
4  
5 void test(int x, int y) {  
6     if (x != y)  
7         if (foo(x) == foo(y))  
8             assert;  
9 }
```

Можно символические
переменные заменить на
конкретные

- Может никогда не остановиться
- Метод полный - если мы достигаем ошибки, то программа достигает её на некоторых данных
- Не надёжный - если анализ останавливается и не нашёл ошибок, то это не значит, что их нет

1 $a = b + c$

```
1 class concolic_int(int):
2     def __new__(cls, val, sym):
3         self =
4             super(concolic_int, cls).__new__(cls, val)
5         self.__val = val
6         self.__sym = sym
7         return self
8     def __add__(self, other):
9         if isinstance(other, concolic_int):
10             value = self.__val + other.__val
11             symbolic = self.__sym + "+" + other.__sym
12         else:
13             value = self.__val + other
14             symbolic = self.__sym + "+" + str(other)
15         return concolic_int(value, symbolic)
```

Как `int` заменить на `concolic_int`?

1 $a = b + c$

1 $a = \text{plus}(b, c)$

```
1 function plus(x, y) {
2     if (x instanceof Concolic) {
3         if (y instanceof Concolic) {
4             return new Concolic(
5                 x._val + y._val,
6                 x._sym + "+" + y._sym
7             );
8         } else {
9             return new Concolic(
10                x._val + y,
11                x._sym + "+" + y.toString()
12            );
13        }
14    } else {
15        ....
16    }
17 }
```

Как кодировать пути?

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

- <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

