Symbolic Execution

Lecture 11: CPEN 400P

Karthik Pattabiraman, UBC

(Slides based on Michael Pradel’s Program Analysis course at Univ. of Stuttgart,
and
Gary Tan’s CSE597: Topics in Software Testing at Penn State Univ.)

Outline

What’s symbolic execution ?

How to perform symbolic execution ?

Symbolic execution in practice

Challenges of symbolic execution

Concolic execution

Symbolic Execution

Reason about behavior of program by ”executing” it with symbolic values

Originally proposed by James King (1976, CACM) and Lori Clarke (1976).

Closely related to idea of abstract interpretation proposed by Cousot and Cousot (1977).

Became practical around 2005 because of advances in constraint solving (SMT solvers)

Uses of Symbolic Execution

General goal: Reason about behavior of program

Basic applications:

- Detect infeasible paths
- Generate test inputs
- Find bugs and vulnerabilities

Advanced applications:

- Generating program invariants
- Prove that two pieces of code are equivalent
- Automated program repair

Example: Is this assertion violated (for any value of a,b,c)?

```
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);
}
```

Let’s try a = 2, b = 3, c = 4

x = -2, y = 0, z = 0   (assertion is true)

What about a = 0, b = 6, c = 1

x = 0, y = 1, z = 2 (assertion is false)

Is there a generic way to determine this ?

Symbolic Execution

Consider variables in the program as inputs

a = a0, b = b0, c = c0

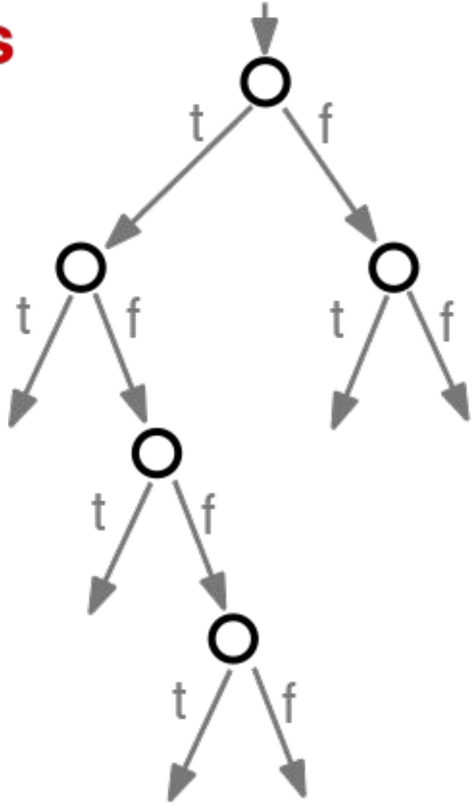
Evaluate every branch symbolically considering both T and F clauses

- Accumulate constraints corresponding to each branch
- Pass the constraints to an SMT solver to determine path feasibility
- Terminate exploration of *infeasible* paths (based on SMT solver's output)

Execution Tree

All possible execution paths

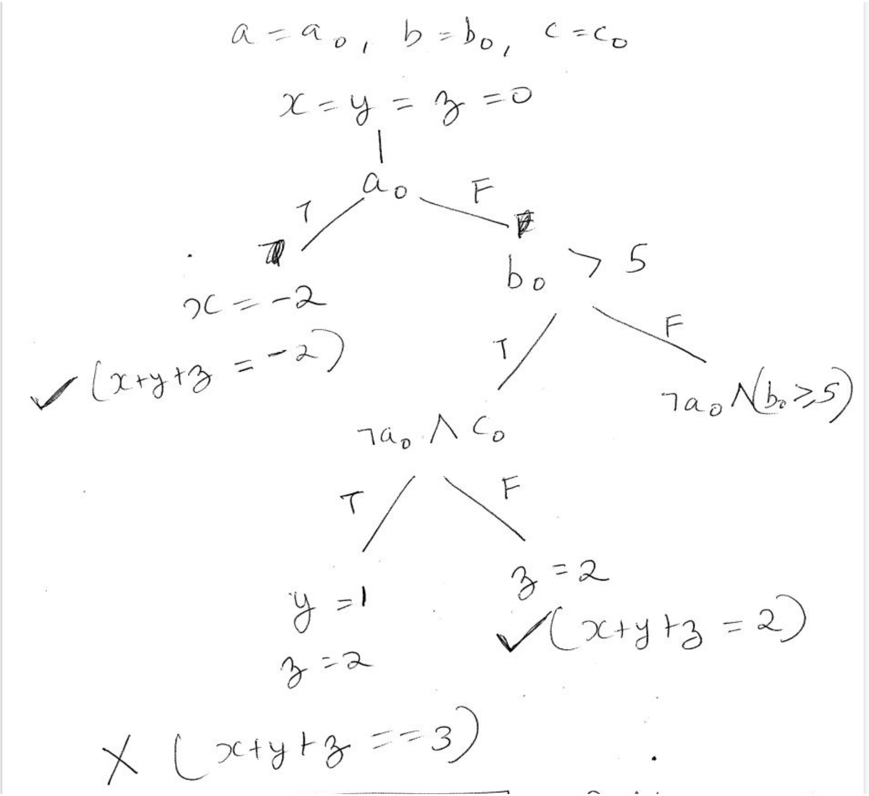
- Binary tree
- Nodes: Conditional statements
- Edges: Execution of sequence on non-conditional statements
- Each path in the tree represents an equivalence class of inputs



Class Activity

Draw the execution tree corresponding to the code shown in Slide 5. Show the paths that satisfy the assertion and those that don’t in the tree.

Class Activity: Solution



Outline

What’s symbolic execution ?

How to perform symbolic execution ?

Challenges of symbolic execution

Concolic execution

Symbolic Values

Unknown values e.g., user inputs are kept symbolically

Symbolic state maps variables to symbolic values

```
function f(x, y) {

    var z = x + y;

    if (z > 0) {

        …
    }
```

Symbolic input: x0, y0

Symbolic state:
z0 = x0 + y0

Path Condition

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

```
function f(x, y) {

    var z = x + y;

    if (z > 0) {

        …
    }
```

Symbolic input: x0, y0

Symbolic state:
z0 = x0 + y0

Path Condition:
x0 + y0 > 0

Satisfiability of Formulas

Determine whether a path is feasible

Check if its path condition is satisfiable

Done by powerful SMT solvers

- SMT stands for Satisfiability Modulo Theory
- Examples: Z3, Yices, STP
- For a satisfiable formula, solvers also provide a concrete solution

Digression: Satisfiability

SAT problem: Given a Boolean formula, is there any value of the variables for which the overall expression evaluates to True ?

- Canonical NP-complete problem (polynomial time to check, exponential time to find solution)
- Modern SAT solvers can find efficiently (with high probability) using heuristics

SMT problem: Generalization of SAT to integer-valued expressions and variables

- NP-hard as opposed to NP complete
- Can be reduced to SAT over multiple bits
- Decidable for some cases only, e.g., Presburger Arithmetic (Natural numbers with addition, but not multiplication

Class Activity: Satisfiable formulas

Which of the following formulas are satisfiable with integer values ? Find a concrete solution if one exists.

1. $a0 + b0 > 1$

2. $(a0 + b0 < 0) \;\&\&\; (a0 - 1 > 5) \;\&\&\; (b0 > 0)$

3. $(a0 + b0 > 1) \;\&\&\; (a0 - 1 < 5) \;\&\&\; (b0 > 0)$

Class Activity

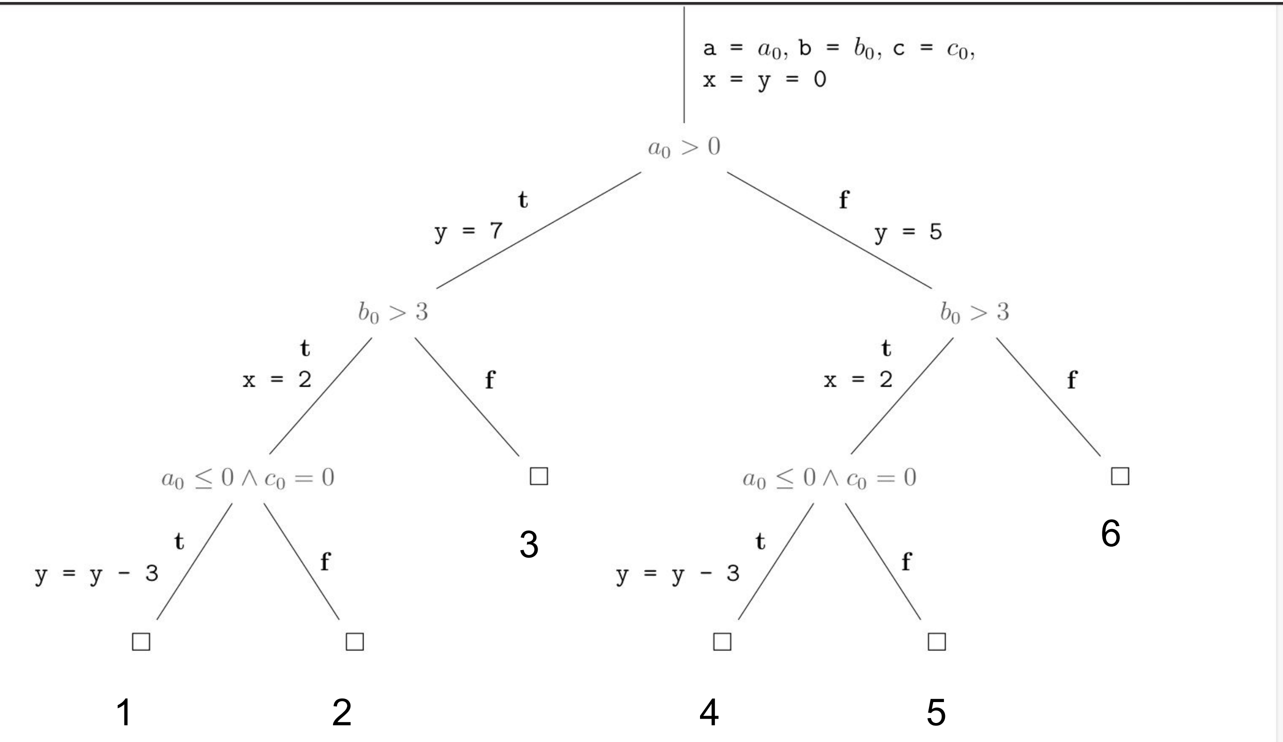
Draw the execution tree for this function corresponding to symbolic execution. Show terminal nodes with a [] in the tree.

Also, write the path conditions corresponding to each path in the tree and solve for them.

Can you come up with values of a, b, and c for which the assertion would fail using the path conditions above ?

```
function foo(a, b, c) {

    let x = y = 0;

    if (a > 0) {
        y = 7;
    } else {
        y = 5;
    }

    if (b > 3) {
        x = 2;

        if (a <= 0 && c = 0) {
            y = y - 3;
        }
    }

    assert( x + y != 5);
}
```

Class Activity: Solution



Class Activity: Solution (contd..)

Path	**Path condition**	**Solution**
1	$a_0 > 0 \land b_0 > 3 \land a_0 \leq 0 \land c_0 = 0$	
2	$a_0 > 0 \land b_0 > 3 \land \neg(a_0 \leq 0 \land c_0 = 0)$	
3	$a_0 > 0 \land \neg(b_0 > 3)$	
4	$\neg(a_0 > 0) \land b_0 > 3 \land a_0 \leq 0 \land c_0 = 0$	
5	$\neg(a_0 > 0) \land b_0 > 3 \land \neg(a_0 \leq 0 \land c_0 = 0)$	
6	$\neg(a_0 > 0) \land \neg(b_0 > 3)$	

Can you solve the above expressions ? Any solution is fine. If none is possible, write UNSAT.

Class Activity: Solution (contd..)

Path	Path condition	Solution (examples)
1	$a0 > 0 \land b0 > 3 \land a0 \le 0 \land c0 = 0$	UNSAT
2	$a0 > 0 \land b0 > 3 \land \neg(a0 \le 0 \land c0 = 0)$	a0 = 1, b0 = 4, co = X
3	$a0 > 0 \land \neg(b0 > 3)$	a0 = 1, b0 = 3, co = X
4	$\neg(a0 > 0) \land b0 > 3 \land a0 \le 0 \land c0 = 0$	a0 = 0, b0 = 4, co = 0
5	$\neg(a0 > 0) \land b0 > 3 \land \neg(a0 \le 0 \land c0 = 0)$	a0 = -1, b0 = 4, co = 1
6	$\neg(a0 > 0) \land \neg(b0 > 3)$	a0 = 0, b0 = 3, co = X

Path condition 6 would lead to the failure of the assert. In this case, x = 0, y = 5, and x+ y = 5

Outline

What’s symbolic execution ?

How to perform symbolic execution ?

Challenges of symbolic execution

Concolic execution

Challenges of Symbolic Execution

Loops and recursion: Infinite execution trees

Path explosion: Number of paths is exponential in the number of conditionals

Environment modeling: Dealing with native/system/library calls

Solver limitations: Dealing with complex path conditions

Heap modeling: Symbolic representation of data structures and pointers

Loops and Recursion

Can lead to path explosion and potentially unbounded number of paths

Two solutions:

1. Come up with loop invariants - fix-points that don’t change no matter how many times you go over the loop. Needs manual intervention
2. Search paths with a bound on how many times you go over them
   a. Doesn’t require manual intervention
   b. Approach adopted by tools such as KLEE or EXE

Dealing with large execution trees

Heuristically choose which branch(es) to explore next

- Random
- Coverage-based
- Distance to “Interesting” program locations (e.g., memory accesses)
- Interleaving symbolic execution with random testing

Environment Modeling

Program behavior may depend on parts of system not analyzed by symbolic execution

Examples are native APIs, interaction with network, file system accesses etc.

```
var fs = require("fs");

var content = fs.readFileSync("/tmp/foo.txt");

if (content === "bar") { ... }
```

Environmental Modeling (contd..)

Solution implemented by KLEE

- If all arguments are concrete, forward to the OS
- Otherwise, provide models that can handle symbolic files
- Goal: Explore all possible legal interactions with the environment

```
var fs = {
    readFileSync: function(file) {
        // doesn’t read actual file system, but
        // models its effects for symbolic file names
        }
    }
```

Solver Limitations

Solvers can’t handle complex or overly large path constraints

- Though state-of-the-art solvers such as Z3 are surprisingly powerful
- Some theories are undecidable (e.g., natural numbers + multiplication)

We won’t cover solvers in this course, but there are many widely available ones

Heap Modeling

Aliasing via pointers

- Complicates symbolic analysis

Treating array elements as one location or separate ones

- Trade-offs in terms of precision and accuracy
- Need to maintain symbolic state for linked data structures

Outline

What’s symbolic execution ?

How to perform symbolic execution ?

Challenges of symbolic execution

Concolic execution

Recall: Fuzz Testing

- Black-box fuzzing
  - Treating the system as a blackbox during fuzzing; not knowing details of the implementation
- Grey-box fuzzing
  - Coverage-based fuzzing (e.g., AFL)
- White-box fuzzing
  - Combines fuzzing with test generation
  - Rather than randomly generating new inputs and hoping that they enable a new path to be executed, compute inputs that will execute a desired path

Solution: Concolic Execution

Concolic = **Con**crete + Symb**olic**

Combining Classical Testing with Automatic Program Analysis

Also called **dynamic symbolic execution**

Program is simultaneously executed with concrete and symbolic inputs

Start off the execution with a random input

The intention is to visit deep into the program execution tree

Concolic execution implementations: SAGE (Microsoft), CREST

Concolic Testing: Main Idea

Mix concrete and symbolic execution = ”concolic”

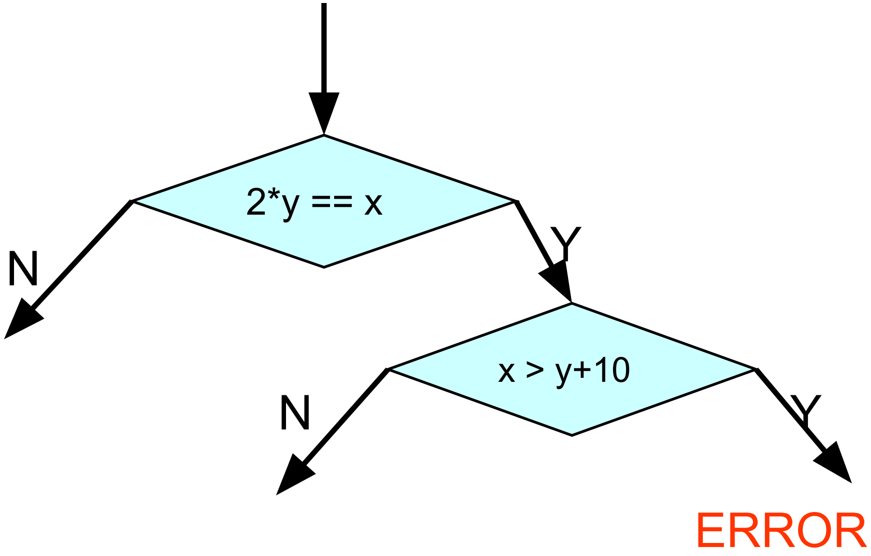
Perform concrete and symbolic execution side-by-side

Gather path constraints while program executes

- After one execution, negate one decision,
- Re-execute with new input that triggers another path

Example

```
void testme (int x, int y)
{
      z = 2*y;
       if (z == x) {
            if (x > y+10) {
                  ERROR;
            }
      }
}
```



Concolic execution example

```
void testme (int x, int y) {

z = 2* y;
if (z == x) {
    if (x > y+10) {
        ERROR;
    }
}
```

	Concrete Execution		Symbolic Execution
concrete state		symbolic state	path condition
x = 22, y = 7		x = a, y = b	

Concolic execution example

```
void testme (int x, int y) {
z = 2* y;

if (z == x) {
    if (x > y+10) {
        ERROR;
    }
}
```

Concrete Execution		Symbolic Execution	
concrete state	symbolic state	path condition	
x = 22, y = 7, z = 14	x = a, y = b, z = 2*b		

Concolic execution example

```
void testme (int x, int y) {
z = 2* y;

if (z == x) {
    if (x > y+10) {
        ERROR;
    }
}
```

Concrete Execution	Symbolic Execution	
concrete state	symbolic state	path condition
x = 22, y = 7, z = 14	x = a, y = b, z = 2*b	

Concolic execution example

```
void testme (int x, int y) {
 z = 2* y;
 if (z == x) {
     if (x > y+10) {
          ERROR;
     }
 }
}
```

Concrete Execution

concrete state

x = 22, y = 7, z = 14

Symbolic Execution

symbolic state

x = a, y = b, z = 2*b

path condition

2*b != a

Concolic execution example

```
void testme (int x, int y) {
 z = 2* y;
 if (z == x) {
     if (x > y+10) {
         ERROR;
     }
 }
}
```

Concrete Execution: concrete state	Symbolic Execution: symbolic state	Symbolic Execution: path condition
		$2*b != a$
$x = 22, y = 7, z = 14$	$x = a, y = b, z = 2*b$	

Solve: $2*b == a$
Solution: $a = 2, b = 1$

Concolic execution example

```
void testme (int x, int y) {

 z = 2* y;
 if (z == x) {
     if (x > y+10) {
          ERROR;
     }
}
```

Concrete Execution: concrete state	Symbolic Execution: symbolic state	path condition
x = 2, y = 1	x = a, y = b	

Concolic execution example

```
void testme (int x, int y) {
  z = 2* y;

  if (z == x) {
    if (x > y+10) {
      ERROR;
    }
}
```

Concrete Execution	Symbolic Execution	
concrete state	symbolic state	path condition
x = 2, y = 1, z = 2	x = a, y = b, z = 2*b	

Concolic execution example

```
void testme (int x, int y) {
 z = 2* y;
 if (z == x) {

     if (x > y+10) {
         ERROR;
     }
}
```

Concrete Execution	Symbolic Execution	
concrete state	symbolic state	path condition
		2*b == a
x = 2, y = 1, z = 2	x = a, y = b, z = 2*b	

Concolic execution example

```
void testme (int x, int y) {
 z = 2* y;
 if (z == x) {

     if (x > y+10) {
          ERROR;
     }
}
```

	Concrete Execution		Symbolic Execution	
	concrete state	symbolic state	path condition
			$2*b == a$
			$a < b + 10$
	$x = 2, y = 1, z = 2$	$x = a, y = b, z = 2*b$	

Concolic execution example

```
void testme (int x, int y) {
 z = 2* y;
 if (z == x) {

    if (x > y+10) {
        ERROR;
    }
}
```

	Concrete Execution	Symbolic Execution	
	concrete state	symbolic state	path condition

Solve: $(2*b == a) \wedge (a - b > 10)$
Solution: $a = 30, b = 15$

concrete state	symbolic state	path condition
x = 2, y = 1, z = 2	x = a, y = b, z = 2*b	2*b == a a - b < 10

Concolic execution example

```
void testme (int x, int y) {
 z = 2* y;

 if (z == x) {
     if (x > y+10) {
          ERROR;
     }
 }
}
```

	Concrete Execution	Symbolic Execution	
	concrete state	symbolic state	path condition
	x = 30, y = 15	x = a, y = b	

Concolic execution example

```
void testme (int x, int y) {

  z = 2* y;
  if (z == x) {

      if (x > y+10) {

          ERROR;
      }
  }

}
```

Concrete Execution

Symbolic Execution

symbolic state

path condition

Program Error

$2*b == a$

$a > b+10$

$x = 30, y = 15$

$z = 30$

$x = a, y = b$

Algorithm

Repeat until all paths are covered

1. Execute program with concrete input i and collect symbolic constraints at branch points: C

2. Negate one constraint to force taking an alternative branch b’: Constraints C’

3. Call constraint solver to find solution for C’ : New concrete input i’

4. Execute with input i’ to take branch b’

5. Check at runtime that b’ is indeed taken. Otherwise: ”divergent execution”

Concolic Execution: Advantages and Disadvantages

When symbolic reasoning is impossible or impractical, fall back to concrete values

- Native/system/API functions
- Operations not handled by solver (e.g., floating point operations)

Disadvantage: Lose some symbolic state and completeness

Large-Scale Concolic Testing - 1

SAGE: Concolic testing tool developed at Microsoft Research

Test robustness against unexpected inputs read from files, e.g.,

- Audio files read by media player
- Office documents read by MS Office

Start with known input files and handle bytes read from files as symbolic input

Use concolic execution to compute variants of these files

Large-Scale Concolic Testing - 2

Applied to hundreds of applications

Over 400 machine years of computation from 2007 to 2012

Found hundreds of bugs, including many security vulnerabilties

One third of all the bugs discovered by file fuzzing during the development of Microsoft's Windows 7

Class Activity

Execute the program using symbolic and concolic execution, with concrete inputs (x = 1 and y = 2), and write the program state for the lines indicated

```
function bar(x, y) {
    // Program state?
    if (x > y) {
        x = 3;
    } else {
        y = 3;
    }
    // Program state?
    y = y * 2;
    // Program state?
    x = baz(x, y);
    // Program state?
    if (x < 0) {
        y = y - 1;
    }
    // Program state?
    assert(x + y == 0);
}
function baz(a, b) { return a - b;  }
```

Class Activity: Solution

Line num	**Concrete Exec.**	**Symbolic Exec.**	**Path Condition**
2			
8			
10			
12			
16			

Class Activity: Solution

Line num	Concrete Exec.	Symbolic Exec.	Path Condition
2	x = 1, y = 2	x = x0, y = y0	N/A
8	x = 1, y = 3	x = x0, y = 3	$\neg(x0 > y0)$
10	x = 1, y = 6	x = x0, y = 6	$\neg(x0 > y0)$
12	x = -5, y = 6	x = x0 - 6, y = 6	$\neg(x0 > y0)$
16	x = -5, y = 5	x = x0 - 6, y = 5	$\neg(x0 > y0) \land (x0 - 6 < 0)$

Class Activity: continued

Can you use the above table to generate a new set of inputs that take the program down a different path from the previous execution, say by negating the branch in line 13, and recompute the symbolic and concolic state with this input.

Does the assertion in line 17 fail as a result of this execution ?

Class Activity: Solution

Can you use the above table to generate a new set of inputs that take the program down a different path from the previous execution.

When we negate the branch in line 13, we get the following path condition:

$\neg(x0 > y0) \land \neg(x0 - 6 < 0)$

Solving for this condition yields one solution as follows (there are infinite solns).

$x0 = 6$

$y0 = 6$

Substituting the above in the concrete and symbolic states yields (next slide)...

Class Activity: Solution

Line num	Concrete Exec.	Symbolic Exec.	Path Condition
2	x = 6, y = 6	x = x0, y = y0	N/A
8	x = 6, y = 3	x = x0, y = 3	$\neg(x0 > y0)$
10	x = 6, y = 6	x = x0 - 6, y = 6	$\neg(x0 > y0)$
12	x = 0, y = 6	x = x0 - 6, y = 6	$\neg(x0 > y0)$
16	x = 0, y = 6	x = x0 - 6, y = 6	$\neg(x0 > y0) \land \neg(x0 - 6 < 0)$

The assertion in line 17: (x + y == 0) will fail due to the above values !

Outline

What’s symbolic execution ?

How to perform symbolic execution ?

Challenges of symbolic execution

Concolic execution