Concolic Execution

Announcements

- Project part 1 - Resubmission deadline Sunday 4/13 at midnight

- Project check in -
  - Lab 4/14 **Group 1**: Khanh Ha, Rachel, Ruth, Ferida, Emily and **Group 2**: Glory, Rebecca, Ally, Megan, Keziah
  - Class 4/16 **Group 3**: Clara, Emma, Alison, Amina, Ranty, Caren and **Group 4** Tianyun, Cecilia C, Cecilia Z, Yang and **Group 5**: Hazel, Sarah, Bridge, Jenny, Reagan

- No class 4/14

Symbolic Execution

- A middle ground between random testing and formal verification
  - Requires less specs than FV
  - Reasons over *all* possible inputs (unlike testing)
- Treats the program's input variables as *symbols* rather than concrete values
- As the program "executes", keep track of the symbolic expression for each value and the path condition to reach each point

Symbolic Execution

```java
public static int test(int x, int y, int z) {

    BufferedReader reader = new BufferedReader(new
FileReader("input.txt"));

    String line = reader.readLine();

    if (x > 0) {
        if (y > x) {
            if (z > y - x) {
                if (x + y + z < 50) {
                    System.out.println("Branch A");
                } else if (x + y - z < 20) {
                    System.out.println("Branch B");
                }
            }
        } else if (y < -x) {
            if (z + y > x) {
                System.out.println("Branch C");
            }
        }
    } else {
        if (x + y + z == 0) {
            System.out.println("Branch D");
        } else if (x + y > z) {
            if (x - y + z < 10) {
                System.out.println("Branch E");
            }
        }
    }

    int sum = 0;
    if (line.length() > 0) {
        for (int i = 0; i < z; i++) {
            if ((y + i) < z) {
                sum += i;
            }
        }

    }
    return z / sum;
```

How many times will this loop execute?
Best case? 0
Worst case? Could be infinite!
Simple programming constructs can make symexc infeasible

Limitations of Symbolic Execution

1. Path Explosion
    a. 2^n paths for each branch (loops and conditions)
2. Infinite Execution Trees
    a. Loops and recursion
    b. Loop unrolling limit
        i. Introduces error
3. SMT solver constraints
    a. Some can only handle LRA
    b. NP-complete problem!

```
int n = Integer.parseInt(args[0]);
int x = 0;
while (x < n) {
	x++;
}
```

Heuristics For Path Prioritization

- Explores paths which are likely to cause assertion errors first
  - Still explores all paths

- read/write dependencies
  - Prioritize paths which write to variables that are read from in the postcondition

Path Prioritization

- How many paths are there?
- Which paths affect assertion values?

```
public class Bank {
    public static void main(String[] args) {
        int x = Integer.parseInt(args[0]); // symbolic
        int y = Integer.parseInt(args[1]); // symbolic

        int balanceA = 100;
        int balanceB = 200;
        int temp = 0;

        if (x > 0) {
            temp = x * 2;
            if (x < 100) {
                balanceA += temp;
            }
        } else {
            temp = 42;
        }

        if (y < 0) {
            balanceB += y;
        }

        assert Math.abs(balanceA - balanceB) <= 300;
    }
}
```

Other Path Prioritization Heuristics

- Explore branches that have deep nested branches first
  - Prioritizes increase in coverage

- Exception oriented search
  - Explore paths that may trigger exceptions first
  - Divisions, field accesses (NPE), array indexes

- Neural approaches!
  - Use ML to decide which heuristics to use
  - Use ML to decide which paths to explore first

Concolic Execution

Concolic Execution

- Concolic = Concrete + Symbolic
  - Hybrid approach of testing and symbolic execution
  - Also called Dynamic Symbolic Execution or DSE

- Program is simultaneously executed with concrete and symbolic inputs
  - Uses concrete values to simplify symbolic constraints

- Solves constraints to guide execution at branch points

Concolic Execution:

```
int foo(int v) {
   return 2*v;
}

void test_me(int x, int y) {
   int z = foo(y);
   if (z == x)
      if (x > y+10)
         ERROR;
}
```

Concrete State	Symbolic State	Path Condition
x = 22, y = 7	x = $\alpha 1$, y = $\alpha 2$	true

Concolic Execution:

```
int foo(int v) {
   return 2*v;
}

void test_me(int x, int y) {
   int z = foo(y);
   if (z == x)
      if (x > y+10)
         ERROR;
}
```

Concrete State	Symbolic State	Path Condition
x = 22, y = 7	x = α1, y = α2	true
z = 14	z = 2*α2	

Concolic Execution:

```
int foo(int v) {
   return 2*v;
}

void test_me(int x, int y) {
   int z = foo(y);
   if (z == x)
      if (x > y+10)
         ERROR;
}
```

Concrete State	Symbolic State	Path Condition
x = 22, y = 7	x = α1, y = α2	true
z = 14	z = 2*α2	

Does z == x? What would we do in traditional SymExc?

Concolic Execution:

```
int foo(int v) {
   return 2*v;
}

void test_me(int x, int y) {
   int z = foo(y);
   if (z == x)
      if (x > y+10)
         ERROR;
}
```

Concrete State	Symbolic State	Path Condition
x = 22, y = 7	x = α1, y = α2	true
z = 14	z = 2*α2	2*α2 != α1

When Concolic Execution hits a branch, we take the branch implied by the concrete values

Concolic Execution - Branching

- Take the branch implied by the concrete values

- When the program is done executing, negate the last path condition and solve for new concrete values
    - Goal: discover new paths
    - Go down each path only once
    - Execute all paths (unlike testing)

- Example: $a \wedge b \wedge c$
    - Next iteration: execute with concrete inputs that satisfy $a \wedge b \wedge !c$

Concolic Execution:

```
int foo(int v) {
   return 2*v;
}

void test_me(int x, int y) {
   int z = foo(y);
   if (z == x)
      if (x > y+10)
         ERROR;
}
```

Concrete State	Symbolic State	Path Condition
x = 22, y = 7	x = α1, y = α2	true
z = 14	z = 2*α2	2*α2 != α1

Solve: 2*α2 == α1
Solution: α1 = 2, α2 = 1

Concolic Execution:

```
int foo(int v) {
   return 2*v;
}

void test_me(int x, int y) {
   int z = foo(y);
   if (z == x)
      if (x > y+10)
         ERROR;
}
```

Concrete State	Symbolic State	Path Condition
x = 2, y = 1	x = α1, y = α2	true

Concolic Execution:

```
int foo(int v) {
   return 2*v;
}

void test_me(int x, int y) {
   int z = foo(y);
   if (z == x)
      if (x > y+10)
         ERROR;
}
```

Concrete State	Symbolic State	Path Condition
x = 2, y = 1	x = α1, y = α2	true
z = 2	z = 2*α2	

Concolic Execution:

```
int foo(int v) {
   return 2*v;
}

void test_me(int x, int y) {
   int z = foo(y);
   if (z == x)
      if (x > y+10)
         ERROR;
}
```

Concrete State	Symbolic State	Path Condition
x = 2, y = 1	x = $\alpha 1$, y = $\alpha 2$	true
z = 2	z = 2*$\alpha 2$	2*$\alpha 2$ == $\alpha 1$

Concolic Execution:

```
int foo(int v) {
   return 2*v;
}

void test_me(int x, int y) {
   int z = foo(y);
   if (z == x)
      if (x > y+10)
         ERROR;
}
```

Concrete State	Symbolic State	Path Condition
x = 2, y = 1	x = $\alpha 1$, y = $\alpha 2$	true
z = 2	z = $2*\alpha 2$	$2*\alpha 2 == \alpha 1$
		$\alpha 1 <= \alpha 2+10$

Concolic Execution:

```
int foo(int v) {
   return 2*v;
}

void test_me(int x, int y) {
   int z = foo(y);
   if (z == x)
      if (x > y+10)
         ERROR;
}
```

Concrete State	Symbolic State	Path Condition
x = 2, y = 1	x = α1, y = α2	true
z = 2	z = 2*α2	2*α2 == α1
		α1 <= α2+10

Solve: (2*α2 == α1) and (α1 <= α2+10)
Solution: α1 = 30, α2 = 15

Concolic Execution:

```
int foo(int v) {
   return 2*v;
}

void test_me(int x, int y) {
   int z = foo(y);
   if (z == x)
      if (x > y+10)
         ERROR;
}
```

Concrete State	Symbolic State	Path Condition
x = 30, y = 15	x = α1, y = α2	true

Concolic Execution:

```
int foo(int v) {
   return 2*v;
}

void test_me(int x, int y) {
   int z = foo(y);
   if (z == x)
      if (x > y+10)
         ERROR;
}
```

Concrete State	Symbolic State	Path Condition
x = 30, y = 15	x = α1, y = α2	true
z = 30	z = 2*α2	

Concolic Execution:

```
int foo(int v) {
   return 2*v;
}

void test_me(int x, int y) {
   int z = foo(y);
   if (z == x)
      if (x > y+10)
         ERROR;
}
```

Concrete State	Symbolic State	Path Condition
x = 30, y = 15	x = α1, y = α2	true
z = 30	z = 2*α2	2*α2 == α1

Concolic Execution:

```
int foo(int v) {
   return 2*v;
}

void test_me(int x, int y) {
   int z = foo(y);
   if (z == x)
      if (x > y+10)
         ERROR;
}
```

Concrete State	Symbolic State	Path Condition
x = 30, y = 15	x = $\alpha 1$, y = $\alpha 2$	true
z = 30	z = 2*$\alpha 2$	2*$\alpha 2$ == $\alpha 1$
		$\alpha 1 > \alpha 2 + 10$

Concolic Execution:

```
int foo(int v) {
   return 2*v;
}

void test_me(int x, int y) {
   int z = foo(y);
   if (z == x)
      if (x > y+10)
         ERROR;
}
```

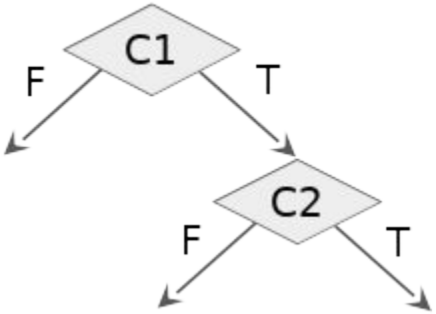
Concrete State	Symbolic State	Path Condition
x = 30, y = 15	x = α1, y = α2	true
z = 30	z = 2*α2	2*α2 == α1
		α1 > α2 + 10

Error!
We explored all 3 input paths with only 3 total concrete states.

Question:

Check all constraints that DSE might possibly solve in exploring the computation tree shown below:

- $C1$
- $C2$
- $\neg C1$
- $\neg C2$
- $C1 \land C2$
- $C1 \land \neg C2$
- $\neg C1 \land C2$
- $\neg C1 \land \neg C2$



A More Complex Example

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

void test_me(int x, int y) {
   int z = foo(y);
   if (z == x)
      if (x > y+10)
         ERROR;
}
```

After first execution, negate the last path condition and solve

Solve: secure_hash(α2) == α1
SAT SOLVER CAN'T SOLVE THIS
Use the concrete state!

A More Complex Example

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

void test_me(int x, int y) {
   int z = foo(y);
   if (z == x)
      if (x > y+10)
         ERROR;
}
```

Solve: secure_hash(7) == $\alpha 1$
Solution: $\alpha 1$ = 601...129 , $\alpha 2$ = 7

Example: Trace the concolic execution

Concrete starting state: x = 1

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

Reminder:
A symbolic value is either a:

- constant (e.g., an integer constant),
- symbol (αi)
- expression formed from αi and constants (α1 + α2, 3α3)

Summary

- Symbolic Execution fallbacks
  - Path explosion and infinite execution paths
  - Calls to SMT solver can be slow

- Concolic Execution
  - Reasons over both concrete and symbolic state
  - Negates the last path condition to explore a new branch at each iteration

- Project reports due sunday night