COSE419: Software Verification

Lecture 3 – Concolic Testing*

Hakjoo Oh 2024 Spring

Motivation

- Writing and maintaining tests is tedious and error-prone
- Idea: Automated Test Generation
 - Generate a regression test suite
 - Execute all reachable statements
 - Catch any assertion violations

Existing Approach 1

- Random Testing
 - Generate random inputs
 - Execute the program on those (concrete) inputs
- Problem
 - Probability of catching error can be astronomically small

Existing Approach 2

- Symbolic Execution
 - Use symbolic values for inputs
 - Execute program symbolically on symbolic input values
 - Collect symbolic path constraints
 - Use theorem prover to check if a branch can be taken
- Problem
 - Incomplete theorem prover
 - Limited scalability

```
int double (int v) {
  return 2*v;
void testme(int x, int y) {
 z := double (y);
  if (z==x) {
  3
   if (x>y+10) {
    4 Crash
    } else { 5 }
```

Execution Tree

1 x: α , y: β pc: true

```
int double (int v) {
  return 2*v;
void testme(int x, int y) {
 z := double (y);
  if (z==x) {
  3
    if (x>y+10) {
    4 Crash
    } else { 5 }
```



```
int double (int v) {
                                                                    \mathbf{x}: \alpha, \mathbf{y}: \beta
                                                                     pc: true
   return 2*v;
void testme(int x, int y) {
                                                                 \mathbf{x}: \alpha, \mathbf{y}: \beta, \mathbf{z}: 2\beta
                                                                     pc: true
   z := double (y);
   if (z==x) {
   3
      if (x>y+10) {
                                                    pc: 2\beta = \alpha
      4 Crash
      } else { 5 }
```

Execution Tree

```
int double (int v) {
                                                                                   \mathbf{x}: \alpha, \mathbf{y}: \beta
                                                                                    pc: true
    return 2*v;
void testme(int x, int y) {
                                                                               \mathbf{x}: \alpha, \mathbf{y}: \beta, \mathbf{z}: 2\beta
                                                                                    pc: true
   z := double (y);
    if (z==x) {
    3
                                                                                             x: \alpha, y: \beta, z: 2\beta pc: 2\beta \neq \alpha
                                                             x: \alpha, y: \beta, z: 2\beta pc: 2\beta = \alpha
       if (x>y+10) {
        4 Crash
       } else { 5 }
```

Execution Tree

```
\mathbf{x}: \alpha, \mathbf{y}: \beta
int double (int v) {
                                                                                                  pc: true
    return 2*v;
void testme(int x, int y) {
                                                                                             \mathbf{x}: \alpha, \mathbf{y}: \beta, \mathbf{z}: 2\beta
                                                                                                  pc: true
    z := double (y);
    if (z==x) {
    3
                                                                       \mathbf{x}: \alpha, \mathbf{y}: \beta, \mathbf{z}: 2\beta
                                                                                                             \mathbf{x}: \alpha, \mathbf{y}: \beta, \mathbf{z}: 2\beta
                                                                          pc: 2\beta = \alpha
                                                                                                                pc: 2\beta \neq \alpha
         if (x>y+10) {
         4 Crash
         } else { 5 }
                                                        \mathbf{x}: \alpha, \mathbf{y}: \beta, \mathbf{z}: 2\beta
                                               pc: 2\beta = \alpha \wedge \alpha > \beta + 10
```

Execution Tree

```
Execution Tree
                                                                                                      \mathbf{x}: \alpha, \mathbf{y}: \beta
int double (int v) {
                                                                                                       pc: true
     return 2*v;
void testme(int x, int y) {
                                                                                                 \mathbf{x}: \alpha, \mathbf{y}: \beta, \mathbf{z}: 2\beta
                                                                                                       pc: true
    z := double (y);
    if (z==x) {
     3
                                                                                                                   \mathbf{x}: \alpha, \mathbf{y}: \beta, \mathbf{z}: 2\beta
                                                                           \mathbf{x}: \alpha, \mathbf{y}: \beta, \mathbf{z}: 2\beta
         if (x>y+10) {
                                                                              pc: 2\beta = \alpha
                                                                                                                      pc: 2\beta \neq \alpha
          4 Crash
         } else { 5 }
                                                          \mathbf{x}: \alpha, \mathbf{y}: \beta, \mathbf{z}: 2\beta
                                                                                                                     \mathbf{x}: \alpha, \mathbf{y}: \beta, \mathbf{z}: 2\beta
                                                                                                     5
                                                  pc: 2\beta = \alpha \wedge \alpha > \beta + 10
                                                                                                            pc: 2\beta = \alpha \land \alpha \le \beta + 10
```

Limitation of Symbolic Execution

```
int foo (int v) {
  return secure_hash(v);
void testme(int x, int y) {
 z := foo (y);
  if (z==x) {
   if (x>y+10) {
     Crash
   } else { }
```

- Approach
 - Store program state concretely and symbolically
 - Solve constraints to guide execution at branch points
 - Explore all execution paths of the unit tested
 - Use concrete values to simplify symbolic constraints
- Example of hybrid analysis
 - Collaboratively combines dynamic and static analysis

```
Symbolic
                                     Concrete
int double (int v) {
                                       State
                                                      State
  return 2*v;
void testme(int x, int y) {
                                                     x=a, y=\beta
                                     x=22, y=7
  z := double (y);
                                                       true
  if (z==x) {
    if (x>y+10) {
      Crash
    } else { }
                         1st iteration
```

```
Concrete
int double (int v) {
                                      State
  return 2*v;
void testme(int x, int y) {
  z := double (y);
                                    x=22, y=7,
                                       z = 14
  if (z==x) {
    if (x>y+10) {
      Crash
    } else { }
```

Symbolic State

$$x=a, y=\beta, z=2*\beta$$
 true

```
Concrete
                                                        Symbolic
int double (int v) {
                                                          State
                                          State
  return 2*v;
void testme(int x, int y) {
  z := double (y);
  if (z==x) {
    if (x>y+10) {
      Crash
    } else { }
                                                      x=a, y=\beta, z=2*\beta
                                       x=22, y=7,
                                          z = 14
                                                        2*\beta \neq a
```

1st iteration

10

```
int double (int v) {
  return 2*v;
void testme(int x, int y) {
  z := double (y);
  if (z==x) {
    if (x>y+10) {
      Crash
    } else { }
```

Concrete State Symbolic State

- Constraint: $2*\beta = \alpha$
- Solution: $a=2,\beta=1$

$$x=22, y=7,$$
 $z=14$

$$x=a, y=\beta, z=2*\beta$$

 $2*\beta \neq a$

```
Symbolic
                                     Concrete
int double (int v) {
                                       State
                                                      State
  return 2*v;
void testme(int x, int y) {
                                                    x=a, y=\beta
                                     x=2, y=1
  z := double (y);
                                                       true
  if (z==x) {
    if (x>y+10) {
      Crash
    } else { }
                        2nd iteration
```

```
Concrete
                                                     Symbolic
int double (int v) {
                                                        State
                                        State
  return 2*v;
void testme(int x, int y) {
  z := double (y);
                                                    x=a, y=\beta, z=2*\beta
                                      x=2, y=1,
                                         z=2
                                                        true
  if (z==x) {
    if (x>y+10) {
      Crash
    } else { }
```

2nd iteration

```
Concrete
int double (int v) {
                                      State
  return 2*v;
void testme(int x, int y) {
  z := double (y);
  if (z==x) {
                                     x=2, y=1,
                                       z=2
    if (x>y+10) {
      Crash
    } else { }
```

Symbolic State

$$x=a, y=\beta, z=2*\beta$$

 $2*\beta = a$

2nd iteration

```
Concrete
                                                         Symbolic
int double (int v) {
                                                           State
                                           State
  return 2*v;
void testme(int x, int y) {
  z := double (y);
  if (z==x) {
    if (x>y+10) {
      Crash
    } else { }
                                                       x=a, y=\beta, z=2*\beta
                                         x=2, y=1,
                                                         2*\beta = a \wedge
                          2nd iteration
                                                          a \le \beta + 10
```

```
Concrete
                                                            Symbolic
int double (int v) {
                                                              State
                                             State
  return 2*v;
void testme(int x, int y) {
                                      - Constraint: 2*\beta = a \land a > \beta+10
  z := double (y);
                                      - Solution: a=30, \beta=15
  if (z==x) {
    if (x>y+10) {
       Crash
    } else { }
                                                          x=a, y=\beta, z=2*\beta
                                           x=2, y=1,
                                                            2*\beta = a \wedge
                           2nd iteration
                                                            a \le \beta + 10
```

```
Symbolic
                                     Concrete
int double (int v) {
                                                      State
                                       State
  return 2*v;
void testme(int x, int y) {
                                                    x=a, y=\beta
                                    x=30, y=15
  z := double (y);
                                                       true
  if (z==x) {
    if (x>y+10) {
      Crash
    } else { }
                        3rd iteration
```

```
Concrete
int double (int v) {
                                      State
  return 2*v;
void testme(int x, int y) {
  z := double (y);
                                    x=30, y=15,
                                       z = 30
  if (z==x) {
    if (x>y+10) {
      Crash
    } else { }
```

Symbolic State

$$x=a, y=\beta, z=2*\beta$$
 true

3rd iteration

```
Concrete
int double (int v) {
                                      State
  return 2*v;
void testme(int x, int y) {
 z := double (y);
  if (z==x) {
                                    x=30, y=15,
                                       z = 30
    if (x>y+10) {
      Crash
    } else { }
```

Symbolic State

$$x=a, y=\beta, z=2*\beta$$

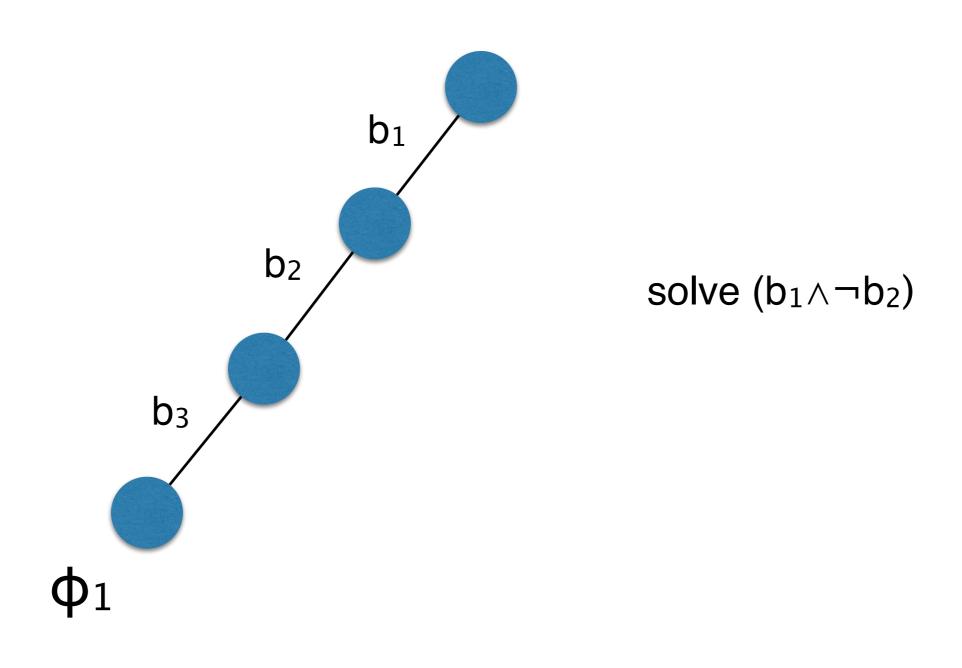
 $2*\beta = a$

3rd iteration

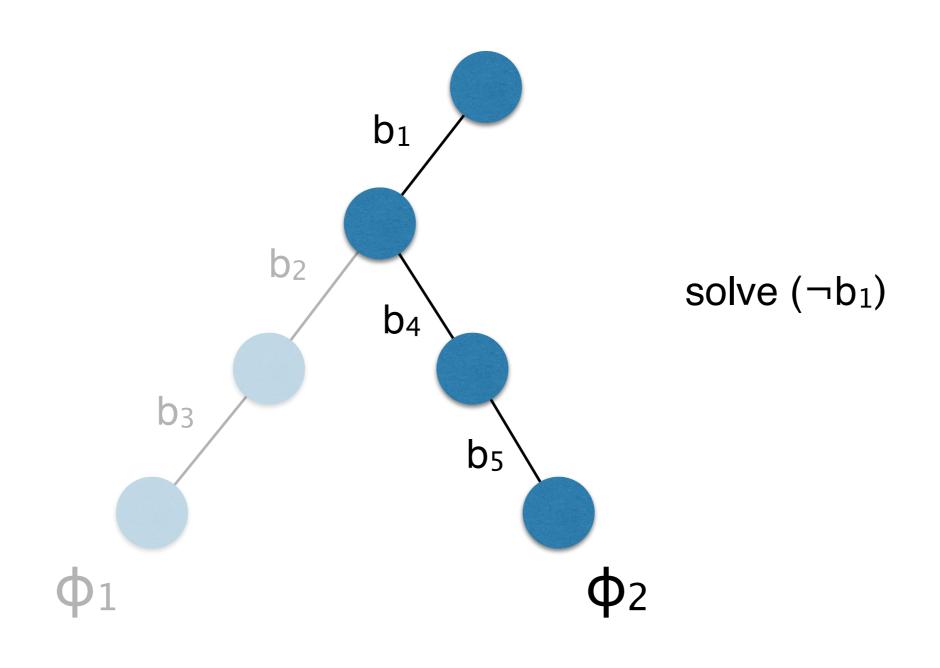
```
Concrete
int double (int v) {
                                       State
  return 2*v;
void testme(int x, int y) {
  z := double (y);
                                 crashing input
  if (z==x) {
    if (x>y+10) {
                                   x=30, y=15,
      Crash ←
                                       z = 30
    } else { }
                        3rd iteration
```

Symbolic State

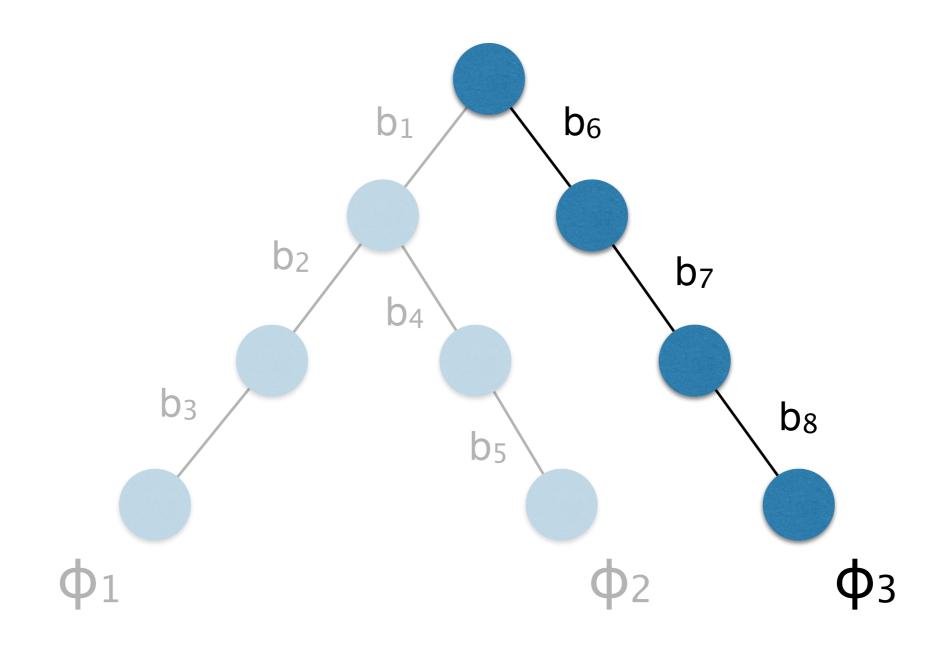
 $x=a, y=\beta, z=2*\beta$ $2*\beta = a \land$ $a > \beta + 10$



execution tree



execution tree



execution tree

Input :Program P, initial input vector v_0 , budget NOutput:The number of branches covered

1: $T \leftarrow \langle \rangle$

```
2: v \leftarrow v_0
 3: for m = 1 to N do
 4: \Phi_m \leftarrow \text{RunProgram}(P, v)
 5: T \leftarrow T \cdot \Phi_m
 6: repeat
 7: (\Phi, \phi_i) \leftarrow \mathsf{Choose}(T) \qquad (\Phi = \phi_1 \land \cdots \land \phi_n)
     until SAT(\bigwedge_{i < i} \phi_i \land \neg \phi_i)
      v \leftarrow \mathsf{model}(\bigwedge_{i < i} \phi_i \land \neg \phi_i)
10: end for
11: return |Branches(T)|
```

11: **return** |Branches(T)|

```
Input: Program P, initial input vector v_0, budget N
Output: The number of branches covered
  1: T \leftarrow \langle \rangle
  2: v \leftarrow v_0
  3: for m = 1 to N do
                                                  Search
  4: \Phi_m \leftarrow \text{RunProgram}(P_m)
  5: T \leftarrow T \cdot \Phi_m
                                                Heuristic
  6: repeat
      (\Phi, \phi_i) \leftarrow \mathsf{Choose}(T) \qquad (\Phi = \phi_1 \wedge \cdots \wedge \phi_n)
       until SAT(\bigwedge_{i < i} \phi_i \land \neg \phi_i)
         v \leftarrow \mathsf{model}(\bigwedge_{j < i} \phi_j \land \neg \phi_i)
 10: end for
```

```
Concrete
                                                     Symbolic
int foo (int v) {
                                                       State
                                       State
  return hash(v);
void testme(int x, int y) {
                                                     x=a, y=\beta
                                     x=22, y=7
  z := foo (y);
                                                       true
  if (z==x) {
    if (x>y+10) {
      Crash
    } else { }
                         1st iteration
```

```
Concrete
                                                      Symbolic
int foo (int v) {
                                                        State
                                        State
  return hash(v);
void testme(int x, int y) {
                                                        x=a, y=\beta,
  z := foo (y);
                                      x=22, y=7,
                                                       z=hash(\beta)
                                     z=601...129
  if (z==x) {
                                                          true
    if (x>y+10) {
      Crash
    } else { }
```

```
Concrete
                                                        Symbolic
int foo (int v) {
                                                          State
                                          State
  return hash(v);
void testme(int x, int y) {
  z := foo (y);
  if (z==x) {
    if (x>y+10) {
      Crash
    } else { }
                                                         x=a, y=\beta,
                                       x=22, y=7,
                                                         z=hash(\beta)
                                      z=601...129
                                                        hash(\beta) \neq a
```

```
int foo (int v) {
  return hash(v);
void testme(int x, int y) {
  z := foo (y);
  if (z==x) {
    if (x>y+10) {
      Crash
    } else { }
```

```
Concrete
State
```

Symbolic State

```
- Constraint: hash(\beta) = \alpha
```

- Replace β by 7: 601...129 = α
- Solution: a = 601...129, $\beta = 7$

```
x=22, y=7,
z=601...129
```

$$x=a, y=\beta,$$

 $z=hash(\beta)$
 $hash(\beta) \neq a$

```
Concrete
                                                     Symbolic
int foo (int v) {
                                                       State
                                       State
  return hash(v);
void testme(int x, int y) {
                                    x=601...129
                                                     x=a, y=\beta
                                        y=7
  z := foo (y);
                                                       true
  if (z==x) {
    if (x>y+10) {
      Crash
    } else { }
```

2nd iteration

```
Concrete
                                                     Symbolic
int foo (int v) {
                                                       State
                                        State
  return hash(v);
void testme(int x, int y) {
                                                       x=a, y=\beta,
                                     x=601...129
  z := foo (y);
                                                       z=hash(\beta)
                                         y=7
  if (z==x) {
                                     z=601...129
                                                          true
    if (x>y+10) {
      Crash
    } else { }
```

2nd iteration

Advantage of Concolic Testing

```
Concrete
                                                      Symbolic
int foo (int v) {
                                                        State
                                         State
  return hash(v);
void testme(int x, int y) {
  z := foo (y);
                                                        x=a, y=\beta,
                                      x=601...129
  if (z==x) {
                                                        z=hash(\beta)
                                          y=7
    if (x>y+10) {
                                      z=601...129
                                                       hash(\beta) = a
      Crash
    } else { }
                         2nd iteration
```

Advantage of Concolic Testing

```
Concrete
                                                        Symbolic
int foo (int v) {
                                                          State
                                          State
  return hash(v);
void testme(int x, int y) {
  z := foo (y);
  if (z==x) {
                                                          x=a, y=\beta,
                                       x=601...129
    if (x>y+10) {
                                                         z=hash(\beta)
                                           y=7
      Crash ←
                                       z=601...129
    } else { }
                                                        hash(\beta) = a \wedge
                                                          a > \beta + 10
                         2nd iteration
```

```
Concrete
                                      State
int foo (int v) {
  return secure_hash(v);
void testme(int x, int y) {
                                   x=22, y=7
  if (x != y) {
    if (foo(x) == foo(y)) {
      Crash
```

Symbolic State

$$x=a, y=\beta$$
 true

1st iteration

```
Concrete
                                                       Symbolic
                                                         State
                                         State
int foo (int v) {
  return secure_hash(v);
void testme(int x, int y) {
  if (x != y) {
                                                       x=a, y=\beta
                                      x=22, y=7
                                                         a \neq \beta
    if (foo(x) == foo(y)) {
      Crash
                          1st iteration
```

```
Concrete
                                      State
int foo (int v) {
  return secure_hash(v);
void testme(int x, int y) {
  if (x != y) {
    if (foo(x) == foo(y)) {
      Crash
                                    x=22, y=7
```

Symbolic State

$$x=\alpha, y=\beta$$

$$\alpha \neq \beta \land$$

$$hash(\alpha) \neq hash(\beta)$$

1st iteration

```
Symbolic
                                               Concrete
                                                                     State
                                                  State
int foo (int v) {
  return secure_hash(v);
                 - Constraint: \alpha \neq \beta \land hash(\alpha) = hash(\beta)
void testme(\frac{1}{2}- Replace a, \beta by 22,7: 22 \neq 7 \wedge 438...861 = 601...129
                 - Unsatisfiable!
  if (x != y
     if (foo(x) == foo(y)) {
        Crash
                                                                    x=a, y=\beta
                                                                      a \neq \beta \wedge
                                               x=22, y=7
```

1st iteration

 $hash(\alpha) \neq hash(\beta)$

```
Symbolic
                                                Concrete
                                                                      State
                                                  State
int foo (int v) {
  return secure_hash(v);
                 - Constraint: \alpha \neq \beta \land hash(\alpha) = hash(\beta)
void testme(\frac{1}{2}- Replace a, \beta by 22,7: 22 \neq 7 \wedge 438...861 = 601...129
                 - Unsatisfiable!
  if (x != y
     if (foo(x) == foo(y)) {
        Crash
                   false negative
                                                                     x=a, y=\beta
                                                                       a \neq \beta \wedge
                                               x=22, y=7
                                                                 hash(a) \neq hash(\beta)
```

1st iteration

```
Symbolic
                                   Concrete
                                     State
                                                    State
void testme(int x) {
                                      x=1
                                                      X=Q
  int A[] = \{ 5, 7, 9 \};
                                                      true
  int i = 0;
  while (i < 3) {
    if (A[i] == x) break;
    i++;
  return i;
                       1st iteration
```

```
Symbolic
                                    Concrete
                                      State
                                                     State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
                                      x=1
                                                       X=Q
  int i = 0;
                                   A = \{5,7,9\}
                                                       true
  while (i < 3) {
    if (A[i] == x) break;
    i++;
  return i;
                       1st iteration
```

```
Symbolic
                                    Concrete
                                      State
                                                      State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
                                     x=1, i=0,
                                                       X=Q
                                    A = \{5,7,9\}
  while (i < 3) {
                                                       true
    if (A[i] == x) break;
    i++;
  return i;
                       1st iteration
```

```
Symbolic
                                        Concrete
                                          State
                                                           State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
  while (i < 3) {
   if (A[i] == x) break;
                                                             X = Q
    i++;
                                                             true
  return i;
                          1st iteration
```

```
Symbolic
                                      Concrete
                                        State
                                                         State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
  while (i < 3) {
    if (A[i] == x) break;
                                       x=1, i=0,

A = \{5,7,9\}
                                                          X = Q
    i++;
                                                          5≠a
  return i;
                         1st iteration
```

```
Symbolic
                                    Concrete
                                      State
                                                     State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
  while (i < 3) {
    if (A[i] == x) break;
                                     x=1, i=1,
    i++;
                                                       X=Q
                                    A = \{5,7,9\}
                                                       5≠a
  return i;
                       1st iteration
```

```
Symbolic
                                        Concrete
                                          State
                                                           State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
  while (i < 3) {
   if (A[i] == x) break;
                                                             X = Q
                                                             5≠a
    i++;
  return i;
                          1st iteration
```

```
Symbolic
                                      Concrete
                                        State
                                                         State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
  while (i < 3) {
    if (A[i] == x) break;
                                        x=1, i=1,
                                                           X=Q
    i++;
                                       A = \{5,7,9\}
                                                       5 \neq a \land 7 \neq a
  return i;
                         1st iteration
```

```
Symbolic
                                      Concrete
                                        State
                                                         State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
  while (i < 3) {
    if (A[i] == x) break;
    i++;
                                        x=1, i=2,
                                                           X=Q
                                       A = \{5,7,9\}
                                                       5 \neq a \land 7 \neq a
  return i;
                         1st iteration
```

```
Symbolic
                                             Concrete
                                               State
                                                                   State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
                                           - X=1, i=2, A = \{5,7,9\}
  while (i < 3) {
   if (A[i] == x) break;
                                                                     X=Q
                                                                 5 \neq a \land 7 \neq a
     i++;
  return i;
```

1st iteration

```
Symbolic
                                       Concrete
                                         State
                                                          State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
  while (i < 3) {
                                                             X=Q
    if (A[i] == x) break;
                                         x=1, i=2,
    i++;
                                        A = \{5,7,9\}
                                                        5 \neq a \land 7 \neq a \land
  return i;
                         1st iteration
```

```
Symbolic
                                         Concrete
                                           State
                                                             State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
  while (i < 3) {
    if (A[i] == x) break;
                                          x=1, i=3,

A = \{5,7,9\}
                                                               X=Q
     i++;
                                                           5 \neq a \land 7 \neq a \land
  return i;
                                                               9≠a
                          1st iteration
```

```
Symbolic
                                       Concrete
                                         State
                                                          State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
  while (i < 3) {
    if (A[i] == x) break;
    i++;
                                         x=1, i=3,
                                                             X=Q
  return i;
                                        A = \{5,7,9\}
                                                        5 \neq a \land 7 \neq a \land
                         1st iteration
```

```
Concrete
                                                           Symbolic
                                            State
                                                              State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
                                     - Constraint: 5 \neq a \land 7 \neq a \land 9 = a
                                     - Solution: a=9
  while (i < 3) {
     if (A[i] == x) break;
     i++;
                                           x=1, i=3,
                                                                X = Q
  return i;
                                          A = \{5,7,9\}
                                                           5 \neq a \land 7 \neq a \land
                           1st iteration
```

```
Symbolic
                                   Concrete
                                     State
                                                    State
void testme(int x) {
                                      x=9
                                                      X=Q
  int A[] = \{ 5, 7, 9 \};
                                                      true
  int i = 0;
  while (i < 3) {
    if (A[i] == x) break;
    i++;
  return i;
```

2nd iteration

```
Symbolic
                                    Concrete
                                      State
                                                      State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
                                       x = 9,
                                                        X=Q
  int i = 0;
                                   A = \{5,7,9\}
                                                        true
  while (i < 3) {
    if (A[i] == x) break;
    i++;
  return i;
```

2nd iteration

```
Symbolic
                                    Concrete
                                      State
                                                     State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
                                     x=9, i=0,
                                                      X=Q
                                    A = \{5,7,9\}
  while (i < 3) {
                                                      true
    if (A[i] == x) break;
    i++;
  return i;
                      2nd iteration
```

```
Symbolic
                                       Concrete
                                          State
                                                           State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
  while (i < 3) {
   if (A[i] == x) break;
                                                            X = Q
    i++;
                                                            true
  return i;
                         2nd iteration
```

```
Symbolic
                                      Concrete
                                        State
                                                         State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
  while (i < 3) {
    if (A[i] == x) break;
                                       x=9, i=0,

A = \{5,7,9\}
                                                          X = Q
    i++;
                                                          5≠a
  return i;
                        2nd iteration
```

```
Symbolic
                                   Concrete
                                     State
                                                     State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
  while (i < 3) {
    if (A[i] == x) break;
                                     x=9, i=1,
    i++;
                                                      X=Q
                                    A = \{5,7,9\}
                                                      5≠a
  return i;
                      2nd iteration
```

```
Symbolic
                                       Concrete
                                          State
                                                           State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
  while (i < 3) {
   if (A[i] == x) break;
                                                            X = Q
                                                            5≠a
    i++;
  return i;
                        2nd iteration
```

```
Symbolic
                                     Concrete
                                        State
                                                        State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
  while (i < 3) {
    if (A[i] == x) break;
                                       x=9, i=1,
                                                          X=Q
    i++;
                                                      5 \neq a \land 7 \neq a
  return i;
                       2nd iteration
```

```
Symbolic
                                      Concrete
                                        State
                                                         State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
  while (i < 3) {
    if (A[i] == x) break;
                                        x=9, i=2,
    i++;
                                                          X=Q
                                       A = \{5,7,9\}
                                                       5 \neq a \land 7 \neq a
  return i;
                       2nd iteration
```

```
Symbolic
                                             Concrete
                                               State
                                                                   State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
                                           - X=9, i=2, A = \{5,7,9\}
  while (i < 3) {
   if (A[i] == x) break;
                                                                     X=Q
                                                                 5 \neq a \land 7 \neq a
     i++;
  return i;
```

2nd iteration

```
Symbolic
                                   Concrete
                                     State
                                                    State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
  while (i < 3) {
    if (A[i] == x) break;
    i++;
                                                      X=Q
                                    x=9, i=2,
  return i;
                                                 5≠a ∧ 7≠a ∧
                                                      9=a
                      2nd iteration
```

```
Concrete
                                                       Symbolic
                                         State
                                                          State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;

    Constraint: 5≠a ∧ 7=a

                                   - Solution: a=7
  while (i < 3) {
    if (A[i] == x) break;
    i++;
                                                            X=Q
                                        x=9, i=2,
  return i;
                                       A = \{5,7,9\}
                                                       5 \neq a \land 7 \neq a \land
                                                            9=a
                        2nd iteration
```

```
Symbolic
                                   Concrete
                                     State
                                                    State
void testme(int x) {
                                     x=7
                                                     X=Q
  int A[] = \{ 5, 7, 9 \};
                                                     true
  int i = 0;
  while (i < 3) {
    if (A[i] == x) break;
    i++;
  return i;
                      3rd iteration
```

```
Symbolic
                                    Concrete
                                      State
                                                      State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
                                       x = 7,
                                                        X=Q
  int i = 0;
                                   A = \{5,7,9\}
                                                        true
  while (i < 3) {
    if (A[i] == x) break;
    i++;
  return i;
```

3rd iteration

```
Symbolic
                                    Concrete
                                      State
                                                     State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
                                     x=7, i=0,
                                                       X=Q
                                    A = \{5,7,9\}
  while (i < 3) {
                                                       true
    if (A[i] == x) break;
    i++;
  return i;
                      3rd iteration
```

```
Symbolic
                                       Concrete
                                         State
                                                          State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
  while (i < 3) {
   if (A[i] == x) break;
                                                            X=Q
    i++;
                                                            true
  return i;
                         3rd iteration
```

```
Symbolic
                                      Concrete
                                        State
                                                         State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
  while (i < 3) {
    if (A[i] == x) break;
                                       x=7, i=0,

A = \{5,7,9\}
                                                          X = Q
    i++;
                                                          5≠a
  return i;
                        3rd iteration
```

```
Symbolic
                                   Concrete
                                      State
                                                     State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
  while (i < 3) {
    if (A[i] == x) break;
                                     x=7, i=1,
    i++;
                                                       X=Q
                                    A = \{5,7,9\}
                                                       5≠a
  return i;
                      3rd iteration
```

```
Symbolic
                                       Concrete
                                          State
                                                           State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
  while (i < 3) {
   if (A[i] == x) break;
                                                             X = Q
                                                            5≠a
    i++;
  return i;
                         3rd iteration
```

```
Symbolic
                                   Concrete
                                     State
                                                    State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
  while (i < 3) {
    if (A[i] == x) break;
    i++;
                                    x=7, i=2,
                                                      X=Q
  return i;
                                                   5≠a ∧ 7=a
                      3rd iteration
```

```
Concrete
                                                   Symbolic
                                      State
                                                     State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
                                - Constraint: 5≠a
                                - Solution: a=5
  while (i < 3) {
    if (A[i] == x) break;
    i++;
                                                       X=Q
                                     x=7, i=2,
  return i;
                                    A = \{5,7,9\}
                                                    5≠a ∧ 7=a
                      3rd iteration
```

```
Symbolic
                                   Concrete
                                     State
                                                    State
void testme(int x) {
                                     x=5
                                                      X=Q
  int A[] = \{ 5, 7, 9 \};
                                                     true
  int i = 0;
  while (i < 3) {
    if (A[i] == x) break;
    i++;
  return i;
                      4th iteration
```

```
Symbolic
                                    Concrete
                                      State
                                                      State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
                                       x = 5,
                                                        X=Q
  int i = 0;
                                   A = \{5,7,9\}
                                                        true
  while (i < 3) {
    if (A[i] == x) break;
    i++;
  return i;
```

4th iteration

```
Symbolic
                                    Concrete
                                      State
                                                      State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
                                     x=5, i=0,
                                                       X=Q
                                    A = \{5, 7, 9\}
  while (i < 3) {
                                                       true
    if (A[i] == x) break;
    i++;
  return i;
                       4th iteration
```

```
Symbolic
                                       Concrete
                                         State
                                                           State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
  while (i < 3) {
   if (A[i] == x) break;
                                                            X=Q
    i++;
                                                            true
  return i;
                         4th iteration
```

```
Symbolic
                                   Concrete
                                     State
                                                    State
void testme(int x) {
  int A[] = \{ 5, 7, 9 \};
  int i = 0;
  while (i < 3) {
    if (A[i] == x) break;
    i++;
                                                      X=Q
  return i;
                                                      5=a
                      4th iteration
```

```
Concrete
                                                    Symbolic
typedef struct cell {
                                                      State
                                       State
  int data;
  struct cell *next;
} cell;
int foo(int v) { return 2*v + 1; }
                                                     x=a, p=\beta
void testme(int x, cell *p) {
                                       x = 236
  if (x > 0)
                                      p=NULL
                                                        true
    if (p != NULL)
      if (foo(x) == p->data)
        if (p->next == p)
          Crash
  return 0;
                         1st iteration
```

```
Concrete
                                                    Symbolic
typedef struct cell {
                                                      State
                                       State
  int data;
  struct cell *next;
} cell;
int foo(int v) { return 2*v + 1; }
void testme(int x, cell *p) {
                                                     x=a, p=\beta
                                       x = 236
  if (x > 0)
    if (p != NULL)
                                      p=NULL
                                                       a > 0
      if (foo(x) == p->data)
        if (p->next == p)
          Crash
  return 0;
                        1st iteration
```

```
Concrete
                                                      Symbolic
typedef struct cell {
                                                        State
                                        State
  int data;
  struct cell *next;
} cell;
int foo(int v) { return 2*v + 1; }
void testme(int x, cell *p) {
  if (x > 0)
    if (p != NULL)
      if (foo(x) == p->data)
        if (p->next == p)
                                                       x=a, p=\beta
          Crash
                                        x = 236
  return 0; ←
                                                        a > 0 \land
                                       p=NULL
                                                       \beta = NULL
```

1st iteration

```
Concrete
                                                        Symbolic
typedef struct cell {
                                                          State
                                          State
  int data;
  struct cell *next;
} cell;
                                - Constraint: a > 0 \land \beta \neq NULL
int foo(int v) { return 2*v - Solution: a = 236, \beta = 634
void testme(int x, cell *p) {
  if (x > 0)
    if (p != NULL)
      if (foo(x) == p->data)
         if (p->next == p)
                                                          x=a, p=\beta
           Crash
                                          x = 236
  return 0; ←
                                                           a > 0 \wedge
                                         p=NULL
                                                          \beta = NULL
                          1st iteration
```

```
Symbolic
                                         Concrete
typedef struct cell {
                                                            State
                                           State
  int data;
  struct cell *next;
} cell;
int foo(int v) { return 2*v + 1; }
                                                            x=a, p=\beta
                                           x = 236
void testme(int x, cell *p) {
                                                           p->data = \gamma
  if (x > 0)^{\blacktriangleleft}
                                           634
                                                 NULL
                                                           p->next = \delta
    if (p != NULL)
       if (foo(x) == p->data)
                                                              true
         if (p->next == p)
           Crash
  return 0;
                           2nd iteration
```

```
Symbolic
                                       Concrete
typedef struct cell {
                                                         State
                                         State
  int data;
  struct cell *next;
} cell;
int foo(int v) { return 2*v + 1; }
void testme(int x, cell *p) {
                                                         x=a, p=\beta
                                        x = 236
  if (x > 0)___
                                                        p->data = \gamma
    if (p != NULL)
                                         634
                                              NULL
                                                        p->next = \delta
      if (foo(x) == p->data)
         if (p->next == p)
                                                           a > 0
           Crash
  return 0;
                         2nd iteration
```

```
Concrete
                                                       Symbolic
typedef struct cell {
                                                         State
                                         State
  int data;
  struct cell *next;
} cell;
int foo(int v) { return 2*v + 1; }
void testme(int x, cell *p) {
  if (x > 0)
                                        x = 236
                                                        x=a, p=\beta
    if (p != NULL)
      if (foo(x) = p->data)
                                                       p->data = \gamma
                                         634
                                              NULL
        if (p->next == p)
                                                       p->next = \delta
           Crash
  return 0;
                                                         a > 0
                                                        \beta \neq NULL
                         2nd iteration
```

```
Symbolic
                                        Concrete
typedef struct cell {
                                                           State
                                           State
  int data;
  struct cell *next;
} cell;
int foo(int v) { return 2*v + 1; }
void testme(int x, cell *p) {
  if (x > 0)
                                                          x=a, p=\beta
    if (p != NULL)
      if (foo(x) == p->data)
                                                         p->data = \gamma
         if (p->next == p)
                                                         p->next = \delta
           Crash
                                         x = 236
  return 0;
                                                            a > 0
                                               NULL
                                         634
                                                          \beta \neq NULL \wedge
                          2nd iteration
                                                          2*a+1 \neq \gamma
```

```
Symbolic
                                          Concrete
typedef struct cell {
                                                              State
                                             State
  int data;
  struct cell *next;
                             - Constraint: a>0 \land \beta \neq NULL \land 2*a+1=\gamma
} cell;
                             - Solution: a = 1, \beta = 3
                                                            NULL
int foo(int v) { return
void testme(int x, cell *p) {
  if (x > 0)
                                                             x=a, p=\beta
    if (p != NULL)
       if (foo(x) == p->data)
                                                            p->data = \gamma
         if (p->next == p)
                                                            p->next = \delta
            Crash
                                           x = 236
  return 0;
                                                               a > 0
                                                 NULL
                                           634
                                                            \beta \neq NULL \wedge
                           2nd iteration
                                                             2*a+1 \neq \gamma
```

```
Symbolic
                                         Concrete
typedef struct cell {
                                                            State
                                           State
  int data;
  struct cell *next;
} cell;
int foo(int v) { return 2*v + 1; }
                                                           x=a, p=\beta
                                            x=1
void testme(int x, cell *p) {
                                                          p->data = \gamma
  if (x > 0)^{\blacktriangleleft}
                                            3
                                                NULL
                                                          p->next = \delta
    if (p != NULL)
       if (foo(x) == p->data)
                                                              true
         if (p->next == p)
           Crash
  return 0;
                           3rd iteration
```

```
Concrete
                                                         Symbolic
typedef struct cell {
                                                           State
                                           State
  int data;
  struct cell *next;
} cell;
int foo(int v) { return 2*v + 1; }
void testme(int x, cell *p) {
                                                          x=a, p=\beta
  if (x > 0)
    if (p != NULL)
                                                         p->data = \gamma
                                           x=1
      if (foo(x) == p->data)
                                                         p->next = \delta
                                           3
                                                NULL
         if (p->next == p)
           Crash
                                                            a > 0 \land
  return 0;
                                                          \beta \neq NULL \wedge
                                                          2*a+1 = \gamma
                          3rd iteration
```

```
Concrete
typedef struct cell {
                                      State
  int data;
  struct cell *next;
} cell;
int foo(int v) { return 2*v + 1; }
void testme(int x, cell *p) {
  if (x > 0)
    if (p != NULL)
      if (foo(x) == p->data)
        if (p->next == p)
          Crash
                                      x=1
  return 0; ←
                                           NULL
                        3rd iteration
```

Symbolic State

 $x=a, p=\beta$ $p->data = \gamma$ $p->next = \delta$ $a>0 \land$ $\beta \neq NULL \land$ $2*a+1 = \gamma \land$ $\delta \neq \beta$

```
Symbolic
                                             Concrete
typedef struct cell {
                                                                 State
                                               State
  int data;
  struct cell *next
} cell;
                        - Constraint: a>0 \land \beta \neq NULL \land 2*a+1=\gamma \land \delta=\beta
int foo(int v) { re-Solution: a = 1, \beta =
                                                           NULL
void testme(int x, cell *p) {
                                                                 x=a, p=\beta
  if (x > 0)
     if (p != NULL)
                                                               p->data = \gamma
       if (foo(x) == p->data)
                                                               p->next = \delta
          if (p->next == p)
            Crash
                                                                  a > 0
                                               x=1
  return 0; ←
                                                                \beta \neq NULL \wedge
                                                    NULL
                                                               2*a+1 = \gamma \land
                             3rd iteration
                                                                   \delta \neq \beta
```

```
Symbolic
                                         Concrete
typedef struct cell {
                                                            State
                                           State
  int data;
  struct cell *next;
} cell;
int foo(int v) { return 2*v + 1; }
                                                            x=a, p=\beta
void testme(int x, cell *p) {
                                            x=1
                                                           p->data = \gamma
  if (x > 0)^{\blacktriangleleft}
                                             3
                                                   NULL
                                                          p->next = \delta
    if (p != NULL)
       if (foo(x) == p->data)
                                                              true
         if (p->next == p)
           Crash
  return 0;
                           4th iteration
```

```
Symbolic
                                          Concrete
typedef struct cell {
                                                             State
                                            State
  int data;
  struct cell *next;
} cell;
int foo(int v) { return 2*v + 1; }
void testme(int x, cell *p) {
                                                             x=a, p=\beta
  if (x > 0)
                                                            p->data=\gamma
    if (p != NULL)
       if (foo(x) == p->data)
                                                            p->next = \delta
                                            x=1
         if (p->next == p)
                                                              a > 0 \land
            Crash ←
                                                   NULL
                                             3
  return 0;
                                                            \beta \neq NULL \wedge
                                                            2*a+1 = \gamma \land
                            4th iteration
                                                               \delta = \beta
```

Summary: Concolic Testing

- An automated, white-box approach to test generation
- Concrete and symbolic execution cooperate w/ each other
 - Concrete execution guides symbolic execution, enabling it to overcome incompleteness of theorem prover
 - Symbolic execution guides generation of concrete inputs, increasing program code coverage
- Further reading:
 - Automatically Generating Search Heuristics for Concolic Testing. ICSE 2018
 - Concolic Testing with Adaptively Changing Search Heuristics. ESEC/FSE 2019
 - SymTuner: Maximizing the Power of Symbolic Execution by Adaptively Tuning External Parameters. ICSE 2022