IS893: Advanced Software Security

11. Symbolic Execution

Kihong Heo

KAIST

Symbolic Execution

- Run programs with symbolic inputs rather than specific inputs
  - Symbol: a class of input
- Compute symbolic memory (mapping from variables to symbolic expressions)
- Symbolic expressions: expressions involving operators over symbols and concrete values
  - E.g., $\alpha \times 4 + \beta$
- Many practical tools: KLEE, Clang static analyzer, etc

Example

```
// Goal: find assert-failing x and y
void foobar(int x, int y) {
  if (x - 1 == 12345) {
    assert(x - y != 0);
  }
}
```

Random testing (concrete execution)

x	x	x - y != 0
125	353	TRUE
45	242156	TRUE
…	…	…

Symbolic execution

$$\mathbf{x} = \alpha_x \wedge \mathbf{y} = \alpha_y \ \wedge \alpha_x - 1 = 12345 \ \wedge \alpha_x - \alpha_y = 0$$

: SAT when $\alpha_x = \alpha_y = 12346$

State

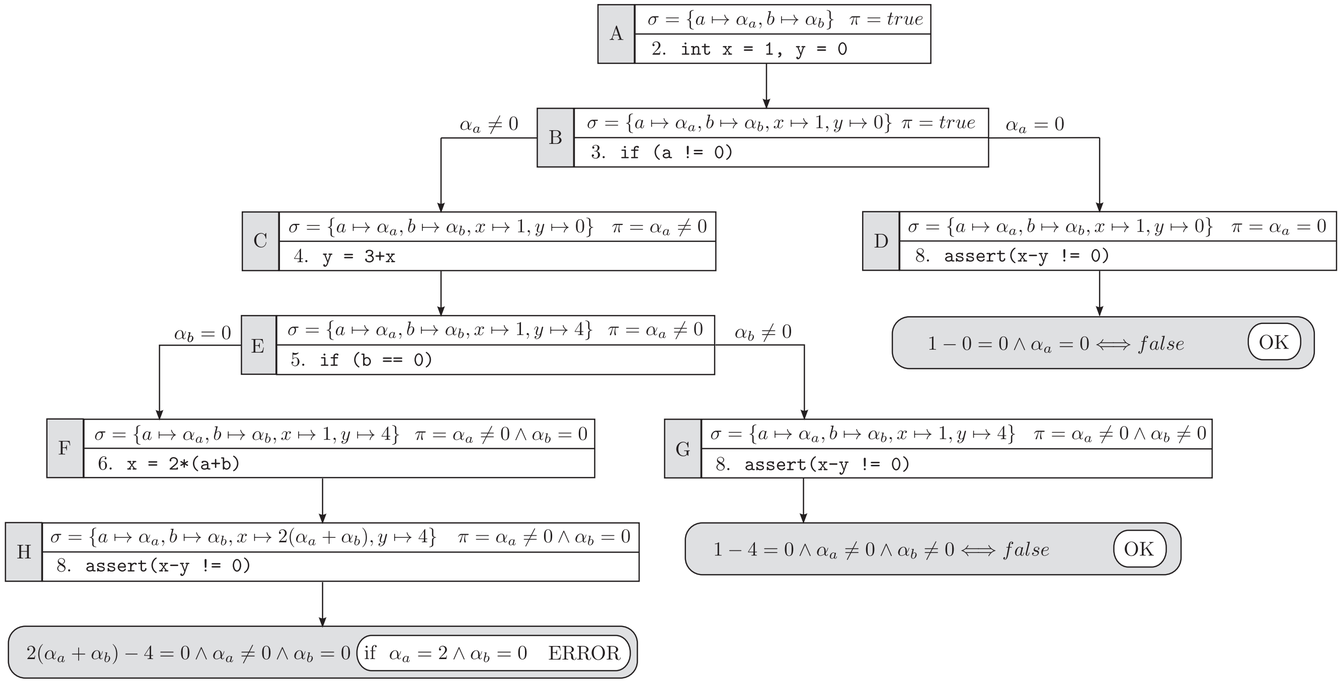
- Symbolic execution maintains a state $(stmt, \sigma, \pi)$ where:
  - $stmt$: the next statement to evaluate
  - σ : a symbolic store (a mapping from program variables to symbolic expressions)
  - π : the path constraint (assumptions on the symbols to reach $stmt$)

State Transition

- Give state $(stmt, \sigma, \pi)$, the symbolic engine changes the state depending on $stmt$:
  - $x = e$: update the symbolic store $\sigma[x \mapsto e_s]$ where e_s is the symbolic expression obtained by evaluating e
  - `if` e `then` s_{true} `else` s_{false} : proceed to each branch with path constraint $\pi_{true} = \pi \wedge e_s$, and $\pi_{false} = \pi \wedge \neg e_s$, respectively
  - `goto` s : advance the symbolic execution to statement s

Example

```
void foobar(int a, int b) {
  int x = 1, y = 0;
  if (a != 0) {
    y = 3 + x;
    if (b == 0)
      x = 2 * (a + b);
  }
  assert(x - y != 0);
}
```



*Baldoni et al., A Survey of Symbolic Execution Techniques, ACM Computing Survey, 2018

Path Feasibility

```
void f(int a, int b) {
  if (a == 12345) {
    assert(b == 0);  // satisfiable?
  }
}
```

```
void f(int a, int b) {
  if (a == 12345) {
    if(b != 0) {
      error();  // reachable?
    }
  }
}
```

- Is a path executed by the symbolic execution engine feasible?
- The feasibility can be determined by an SMT solver
  - If feasible, crashing inputs are found
  - If not, it is a spurious path

Theoretical Aspects

- Symbols soundly subsumes all possibilities of concrete values
- SMT solvers find failing inputs
- May not terminate in the presence of loops
  - Some stopping criteria needed (e.g., bounded loop iterations)

Practical Limitation: Scalability

- State space explosion: N branches → 2^N paths
  - Many of them are infeasible paths
- Constraint solving: limited scalability and expressiveness
  - E.g., large constraints, non-linear constraints, etc

```
void foobar(int x, int y, int z) {
  if (pow(x, 3) + pow(y, 3) == pow(z, 3)) {
    error();
  } else {
    ok();
  }
}
```

Dynamic Symbolic Execution

- Combine concrete execution with symbolic execution
  - testing (scalability) + symbolic execution (constraint solving)
- Execute the program with random concrete inputs
- Keep track of both concrete values and symbolic constraints
- Use concrete values to simplify symbolic constraints

Example

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

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

Concrete State	Symbolic State	Path Constraint
x = 22 y = 7	x = α_x y = α_y	true

1st iteration

Example

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

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

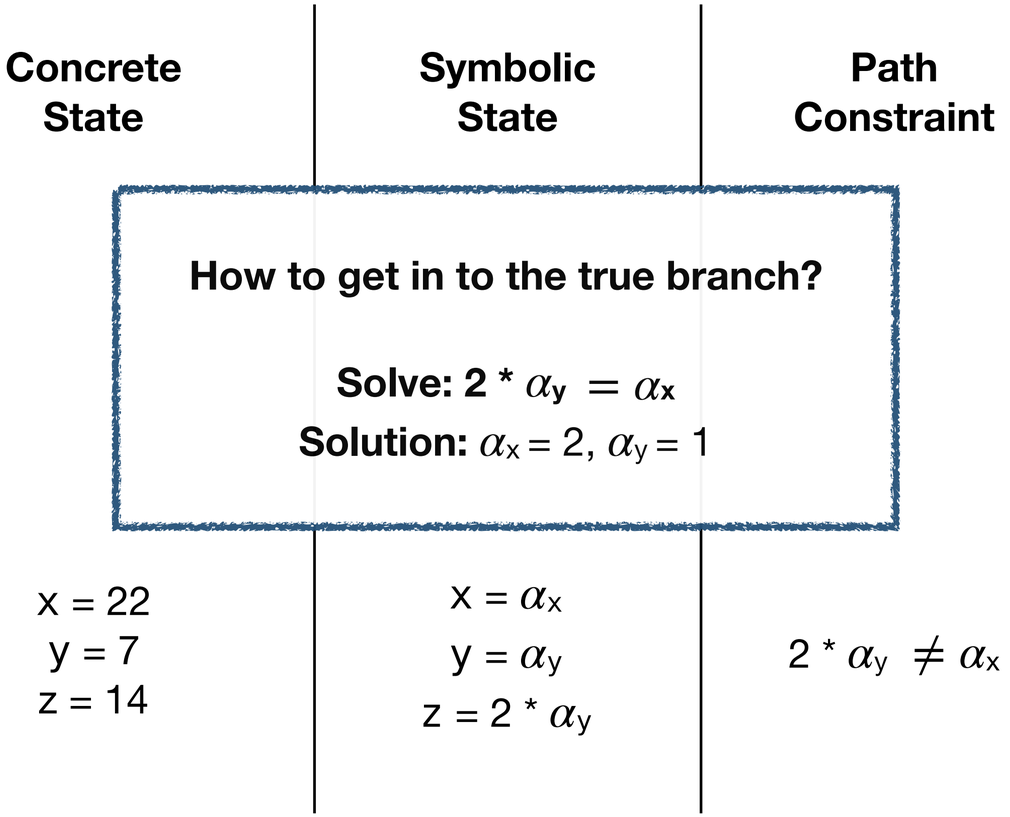
Concrete State	Symbolic State	Path Constraint
x = 22 y = 7 z = 14	x = α_x y = α_y z = 2 * α_y	true

1st iteration

Example

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

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



1st iteration

Example

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

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

Concrete State	Symbolic State	Path Constraint
x = 2 y = 1	x = α_x y = α_y	true

2nd iteration

Example

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

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

Concrete State	Symbolic State	Path Constraint
x = 2 y = 1 z = 2	x = α_x y = α_y z = 2 * α_y	true

2nd iteration

Example

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

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

Concrete State	Symbolic State	Path Constraint
x = 2 y = 1 z = 2	x = α_x y = α_y z = 2 * α_y	2 * α_y = α_x

2nd iteration

Example

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

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

Concrete State	Symbolic State	Path Constraint
x = 2 y = 1 z = 2	x = α_x y = α_y z = 2 * α_y	2 * α_y = α_x $\wedge$ $\alpha_x \leq \alpha_y + 10$

How to get in to the true branch?

Solve: $2 * \alpha_y = \alpha_x \wedge \alpha_x > \alpha_y + 10$

Solution: $\alpha_x = 30, \alpha_y = 15$

2nd iteration

Example

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

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

Concrete State	Symbolic State	Path Constraint
x = 30 y = 15	x = α_x y = α_y	π = true

3rd iteration

Example

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

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

Concrete State	Symbolic State	Path Constraint
x = 30 y = 15 z = 30	x = α_x y = α_y z = 2 * α_y	π = true

3rd iteration

Example

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

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

Concrete State	Symbolic State	Path Constraint
x = 30 y = 15 z = 30	x = α_x y = α_y z = 2 * α_y	2 * α_y = α_x

3rd iteration

Example

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

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

Concrete State	Symbolic State	Path Constraint
x = 30 y = 15 z = 30	x = α_x y = α_y z = 2 * α_y	2 * α_y = α_x $\wedge$ α_x > α_y + 10

3rd iteration

More Complex Example

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

Concrete State	Symbolic State	Path Constraint
x = 22 y = 7	x = α_x y = α_y	true

1st iteration

More Complex Example

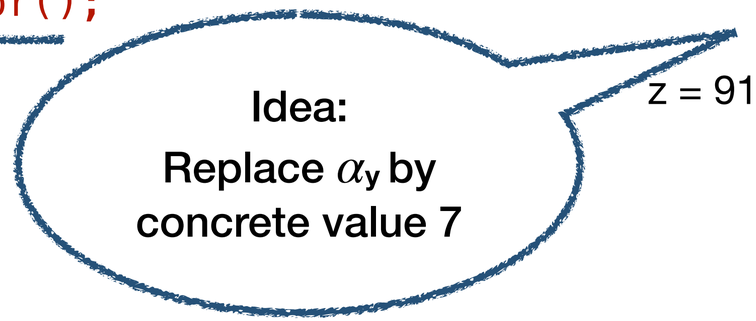
```
void test_me(int x, int y) {
  int z = secure_hash(y);
  if (z == x)
    if (x > y + 10)
      error();
}
```

Concrete State	Symbolic State	Path Constraint
x = 22 y = 7 z = 912490858127853	x = α_x y = α_y z = secure_hash(α_y)	true

1st iteration

More Complex Example

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



Concrete State	Symbolic State	Path Constraint
x = 22 y = 7 z = 912490858127853	x = α_x y = α_y z = secure_hash(α_y)	secure_hash(α_y) $\neq \alpha_x$

How to get in to the true branch?

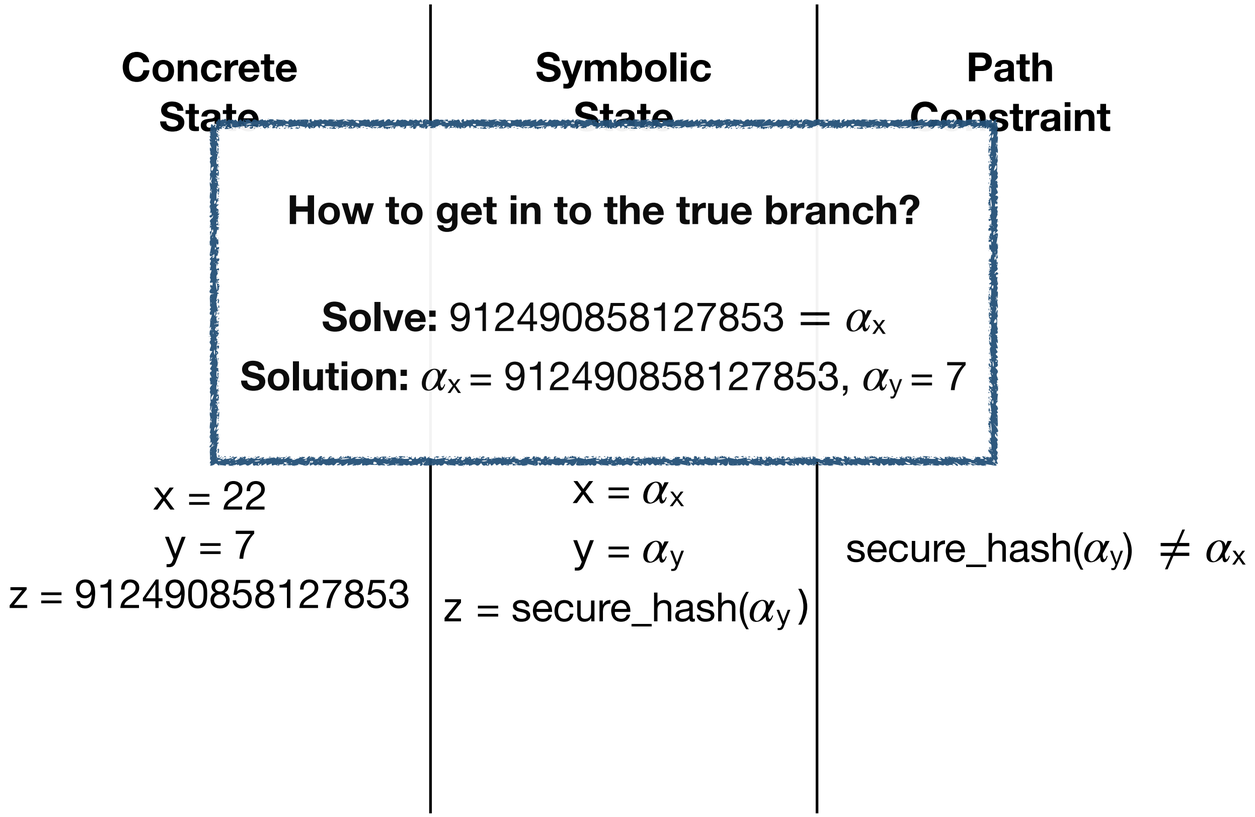
Solve: secure_hash(α_y) $= \alpha_x$

Extremely hard to solve!

1st iteration

More Complex Example

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



1st iteration

More Complex Example

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

Concrete State	Symbolic State	Path Constraint
x = 912490858127853 y = 7	x = α_x y = α_y	true

2nd iteration

More Complex Example

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

Concrete State	Symbolic State	Path Constraint
x = 912490858127853 y = 7 z = 912490858127853	x = α_x y = α_y z = secure_hash(α_y)	true

2nd iteration

More Complex Example

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

Concrete State	Symbolic State	Path Constraint
x = 912490858127853 y = 7 z = 912490858127853	x = α_x y = α_y z = secure_hash(α_y)	secure_hash(α_y) = α_x

2nd iteration

More Complex Example

```
void test_me(int x, int y)
  int z = secure_hash(y);
  if (z == x)
    if (x > y + 10)
      error();
}
```

Concrete State	Symbolic State	Path Constraint
x = 912490858127853 y = 7 z = 912490858127853	x = α_x y = α_y z = secure_hash(α_y)	secure_hash(α_y) = α_x $\wedge$ $\alpha_x > \alpha_y + 10$

2nd iteration

Characteristics

- Reduced number of calls to SMT solvers
  - If reachable by the concrete execution, then no need for constraint solving
- Simplified path constraints by replacing symbols with concrete values
- Caveat: may miss real bugs due to the concretization
  - Unsafely declare a reachable branch as unreachable

Catastrophic Example

```
void test_me(int x, int y) {
 if (x != y) {
    int z = secure_hash(x);
    int w = secure_hash(y);
    if (z == w)
      error();
  }
}
```

Concrete State	Symbolic State	Path Constraint
x = 22 y = 7	x = α_x y = α_y	true

1st iteration

Catastrophic Example

```
void test_me(int x, int y) {
 if (x != y) {
    int z = secure_hash(x);
    int w = secure_hash(y);
    if (z == w)
      error();
  }
}
```

Concrete State	Symbolic State	Path Constraint
x = 22 y = 7 z = 124912759192859 w = 912490858127853	x = α_x y = α_y	$\alpha_x \neq \alpha_y$

1st iteration

Catastrophic Example

```
void test_me(int x, int y) {
 if (x != y) {
    int z = secure_hash(x);
    int w = secure_hash(y);
    if (z == w)
      error();
  }
}
```

Concrete State	Symbolic State	Path Constraint
x = 22 y = 7 z = 124912759192859 w = 912490858127853	x = α_x y = α_y	$\alpha_x \neq \alpha_y$ $\wedge$ hash(α_x) $\neq$ hash(α_y)

How to get in to the true branch?

Solve: $\alpha_x \neq \alpha_y \wedge$ hash(α_x) = hash(α_y)

Use concrete vales!

1st iteration

Catastrophic Example

```
void test_me(int x, int y) {
 if (x != y) {
    int z = secure_hash(x);
    int w = secure_hash(y);
    if (z == w)
      error();
  }
}
```

Concrete State	Symbolic State	Path Constraint
x = 22 y = 7 z = 124912759192859 w = 912490858127853	x = α_x y = α_y	$\alpha_x \neq \alpha_y$ $\wedge$ hash(α_x) $\neq$ hash(α_y)

How to get in to the true branch?

Solve: $22 \neq 7 \wedge 124\ldots859 = 912\ldots853$

Unsatisfiable

1st iteration

Conclusion

- Symbolic execution: run programs with symbolic inputs
  - Effective exploration compared to random testing (i.e., white box)
  - Less scalable because of constraint solving
- Dynamic symbolic execution: combine concrete and symbolic execution
  - Efficient: less number of calls to SMT solver, simplified constraints
  - False negative: may miss real bugs