Symbolic Execution

Sergey Mechtaev

mechtaev@pku.edu.cn

Peking University

Propositional Logic

- Boolean variables: $A, B, C,$ ...
- Logical signs:
  - $\wedge$ – and
  - $\vee$ – or
  - $\neg$ – not
- Propositional formulas:
  - A
  - $A \wedge B$
  - $(A \vee B) \wedge \neg C$

Satisfiability Problem (SAT)

Definition. The problem of determining whether the variables of a given propositional formula can be consistently replaced by the values TRUE or FALSE in such a way that the formula evaluates to TRUE.

- Satisfiable formula:
  - $(A \lor B) \land \neg C$
  - Assignment: $A \mapsto \mathrm{TRUE}, B \mapsto \mathrm{FALSE}, C \mapsto \mathrm{FALSE}$
- Unsatisfiable formula:
  - $A \land \neg A$
- NP-complete, but practical solutions exist

Satisfiability Modulo Theories (SMT)

- Extending propositional logic using theories
- Linear integer arithmetic (LIA):
  - $(A > B \vee B > C) \wedge \neg(A + C = 5)$
- Linear arithmetic over the rationals (LRA):
  - $A = 3.14 \times B + C \wedge B > C$
- Bitvectors (BV):
  - $A[15:0] = (B[15:8] :: C[7:0]) \ll D[3:0]$
- Arrays (AR):
  - $\mathrm{select}(\mathrm{store}(A, 0, 10), 0) = 10$

Theory of Bitvectors

- Operations over Bitvectors (sequences of bits) that emulate computer hardware
- Bitvector versions of arithmetic operators:
  - bvadd – addition
  - bvmul – multiplication
  - Etc.
- Concatenation:
  - $\text{cancat}(\#b0010, \#b1110) = \#b00101110$
- Extraction:
  - $\text{extract}(\#b00101110, 7, 4) = \#b0010$

Theory of Arrays

- $\mathrm{store}(A, i, x)$ – array obtained from A by replacing the element at position i with value x
- $\mathrm{select}(A, i)$ – element stored in array A at position i
- Axiom:

$$i = j \Rightarrow select(store(A, i, x), j) = x$$

Examples of Formulas

- $(A \vee B) \wedge (\neg A \vee \neg B)$
- $(A > B) \wedge (B > 5) \wedge (5 > A)$
- $(A > B \vee B > 5) \wedge (5 > A)$
- $\text{select}(\text{store}(A, 0,5), 0) = 6$

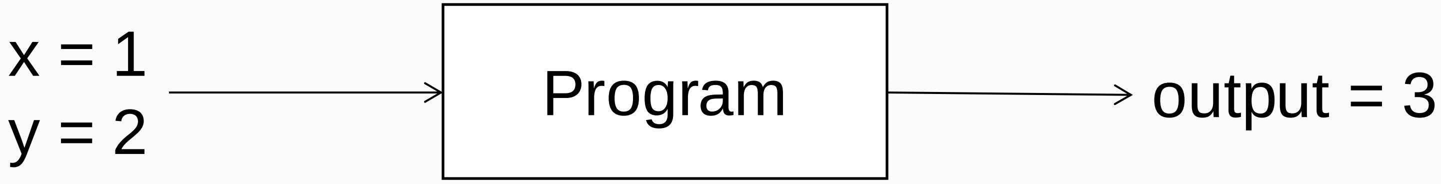
Microsoft Z3 Solver

- One of the most efficient SMT solvers
- Supports various theories:
  - LIA, LRA, BV, AR, …
- Many applications:
  - Program analysis
  - Software verification
  - Program synthesis
  - Etc

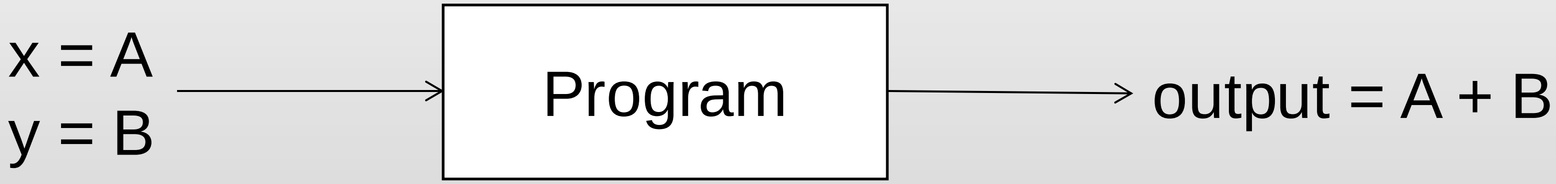
https://github.com/Z3Prover

Symbolic Execution

- Concrete execution:



- Symbolic execution (with **symbolic variables**, instead of concrete values)



Symbolic Execution Example

Program

```
int foo(int x, int y) {
    int t = y;
    if (x > y)
        t = x;
    if (t == 4) {
        return x;
    } else {
        return x + y;
    }
}
```

State

$x = A$

$y = B$

$t = \text{undefined}$

$\text{output} = \text{undefined}$

Path condition

$True$

Symbolic Execution Example

Program

```
int foo(int x, int y) {
    int t = y;
    if (x > y)
        t = x;
    if (t == 4) {
        return x;
    } else {
        return x + y;
    }
}
```

State

$x = A$

$y = B$

$t = \mathrm{B}$

$\mathrm{output} = \mathrm{undefined}$

Path condition

$True$

Symbolic Execution Example

Program

```
int foo(int x, int y) {
    int t = y;
    if (x > y)
        t = x;
    if (t == 4) {
        return x;
    } else {
        return x + y;
    }
}
```

branch

State

$x = A$
$y = B$
$t = \mathrm{B}$
$\mathrm{output} = \mathrm{undefined}$

Path condition

$True$

Symbolic Execution Example

Program

```
int foo(int x, int y) {
    int t = y;
    if (x > y)
        t = x;
    if (t == 4) {
        return x;
    } else {
        return x + y;
    }
}
```

State

$x = A$
$y = B$
$t = A$
$\text{output} = \text{undefined}$

Path condition

$A > B$

Symbolic Execution Example

Program

```
int foo(int x, int y) {
    int t = y;
    if (x > y)
        t = x;
    if (t == 4) {
        return x;
    } else {
        return x + y;
    }
}
```

branch

State

$x = A$
$y = B$
$t = \mathrm{A}$
$\mathrm{output} = \mathrm{undefined}$

Path condition

$A > B$

Symbolic Execution Example

Program

```
int foo(int x, int y) {
    int t = y;
    if (x > y)
        t = x;
    if (t == 4) {
        return x;
    } else {
        return x + y;
    }
}
```

State

$x = A$
$y = B$
$t = \mathrm{A}$
$\mathrm{output} = \mathrm{A}$

Path condition

$A > B$
$A = 4$

Symbolic Execution Example

Program

```
int foo(int x, int y) {
    int t = y;
    if (x > y)
        t = x;
    if (t == 4) {
        return x;
    } else {
        return x + y;
    }
}
```

branch

State

$x = A$
$y = B$
$t = \mathrm{A}$
$\mathrm{output} = \mathrm{undefined}$

Path condition

$A > B$

Symbolic Execution Example

Program

```
int foo(int x, int y) {
    int t = y;
    if (x > y)
        t = x;
    if (t == 4) {
        return x;
    } else {
        return x + y;
    }
}
```

State

$x = A$
$y = B$
$t = A$
$\text{output} = A + B$

Path condition

$A > B$
$A \neq 4$

Symbolic Execution Example

Program

```
int foo(int x, int y) {
    int t = y;
    if (x > y)
        t = x;
    if (t == 4) {
        return x;
    } else {
        return x + y;
    }
}
```

branch

State

$x = A$
$y = B$
$t = \mathrm{B}$
$\mathrm{output} = \mathrm{undefined}$

Path condition

$True$

Symbolic Execution Example

Program

```
int foo(int x, int y) {
    int t = y;
    if (x > y)
        t = x;
    if (t == 4) {
        return x;
    } else {
        return x + y;
    }
}
```

branch

State

$x = A$

$y = B$

$t = \mathrm{B}$

$\mathrm{output} = \mathrm{undefined}$

Path condition

$A \leq B$

Symbolic Execution Example

Program

```
int foo(int x, int y) {
    int t = y;
    if (x > y)
        t = x;
    if (t == 4) {
        return x;
    } else {
        return x + y;
    }
}
```

State

$x = A$
$y = B$
$t = \mathrm{B}$
$\mathrm{output} = \mathrm{A}$

Path condition

$A \leq B$
$B = 4$

Symbolic Execution Example

Program

```
int foo(int x, int y) {
    int t = y;
    if (x > y)
        t = x;
    if (t == 4) {
        return x;
    } else {
        return x + y;
    }
}
```

branch

State

$x = A$

$y = B$

$t = \mathrm{B}$

$\mathrm{output} = \mathrm{undefined}$

Path condition

$A \leq B$

Symbolic Execution Example

Program

```
int foo(int x, int y) {
    int t = y;
    if (x > y)
        t = x;
    if (t == 4) {
        return x;
    } else {
        return x + y;
    }
}
```

State

$x = A$
$y = B$
$t = \mathrm{B}$
$\mathrm{output} = \mathrm{A} + \mathrm{B}$

Path condition

$A \leq B$
$B \neq 4$

Symbolic Execution Example

Program

```
int foo(int x, int y) {
    int t = y;
    if (x > y)
        t = x;
    if (t == 4) {
        return x;
    } else {
        return x + y;
    }
}
```

Summary

- $A > B \wedge A = 4$
  $\text{output} = \mathrm{A}$
- $A > B \wedge A \neq 4$
  $\text{output} = \mathrm{A} + \mathrm{B}$
- $A \leq B \wedge B = 4$
  $\text{output} = \mathrm{A}$
- $A \leq B \wedge B \neq 4$
  $\text{output} = \mathrm{A} + \mathrm{B}$

Infeasible Paths

```
...
if (x > 0)
    x++;
if (x < 2)
    x--;
...
```

Path condition:

$A > 0$

$A + 1 < 2$

SMT Solver

Unsatisfiable

Applications of Symbolic Execution

- Test generation
  - By solving path conditions, can generate test inputs that provide high path coverage
- Bug finding/vulnerability detection
- Software verification
- Reverse engineering
- Debugging
- Etc

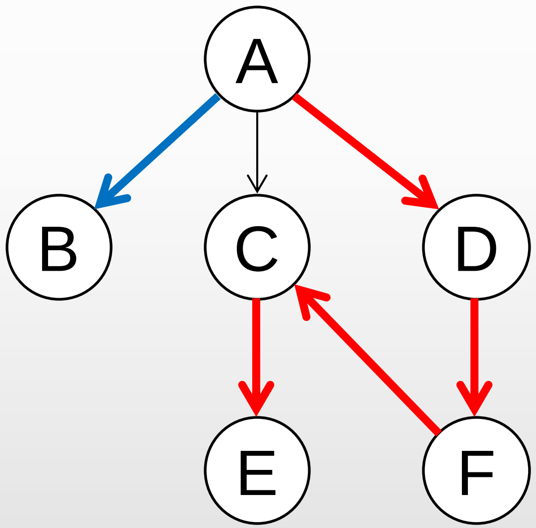
Path Explosion Problem

Programs have infinite number of paths:

```
int foo(int x) {
    while (x > 0)
        x--;
    ...
}
```

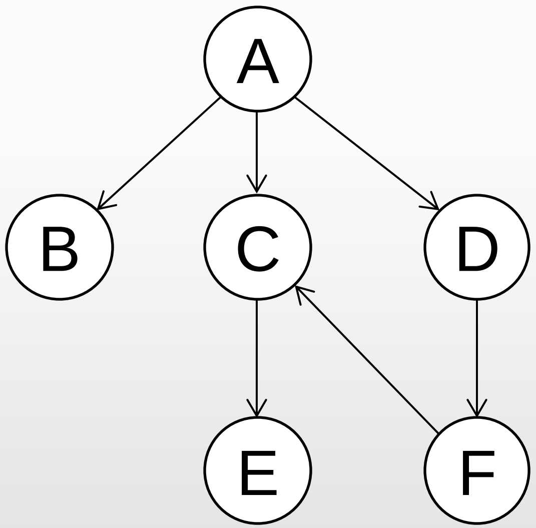
Search Heuristics

Depth-first search



ABDFCE

Breadth-first search



ABCDEF

Constraint Solving Limitations

Cannot solve complex constraints:

```
if (hash(pass) = "553AE8C9...") {
    ...
}
```

Environment Interactions

- Many programs are controlled by environment input – e.g., command-line arguments, files, environment variables, keyboard, network
- Would like to explore all interactions with environment
- Code is not available