SOFT620020.01
Advanced Software Engineering

Bihuan Chen, Pre-Tenure Assoc. Prof.

bhchen@fudan.edu.cn

https://chenbihuan.github.io

Course Outline

Date	Topic
Sep. 09	Introduction
Sep. 16	Testing Overview
Sep. 23	Guided Random Testing
Sep. 30	Search-Based Testing
Oct. 12	Performance Analysis
Oct. 14	Presentation 1
Oct. 21	Security Testing
Oct. 28	Compiler Testing

Date	Topic
Nov. 04	Mobile Testing
Nov. 11	Delta Debugging
Nov. 18	Bug Localization
Nov. 25	Presentation 2
Dec. 02	Automatic Repair
Dec. 09	**Symbolic Execution**
Dec. 16	Big Code Analysis
Dec. 23	Presentation 3

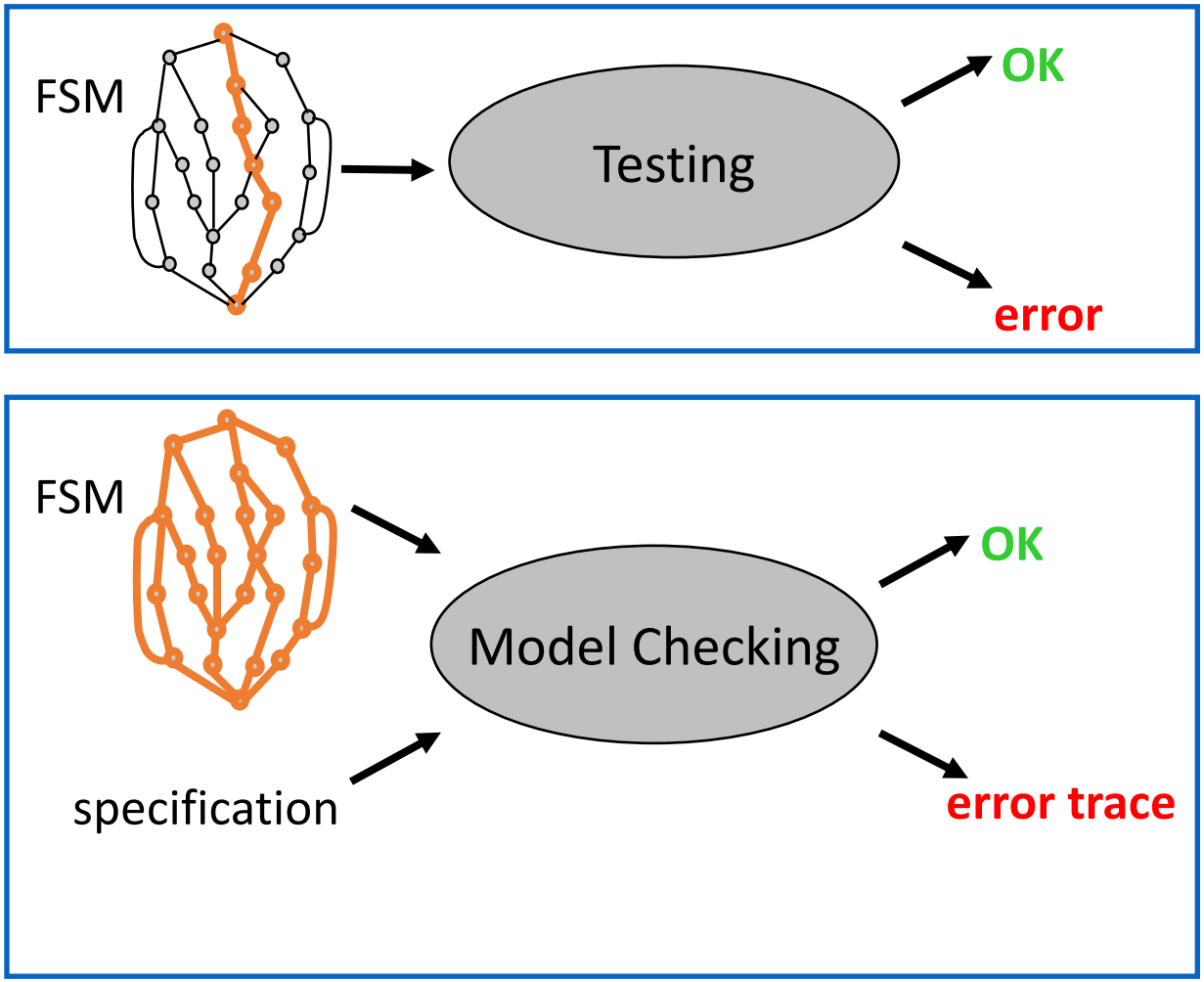
Discussion – Will the Assertion Fail?

```
0. int x, y;
1. if (x>0) {
2.         assert(x>=0);
3. }
```

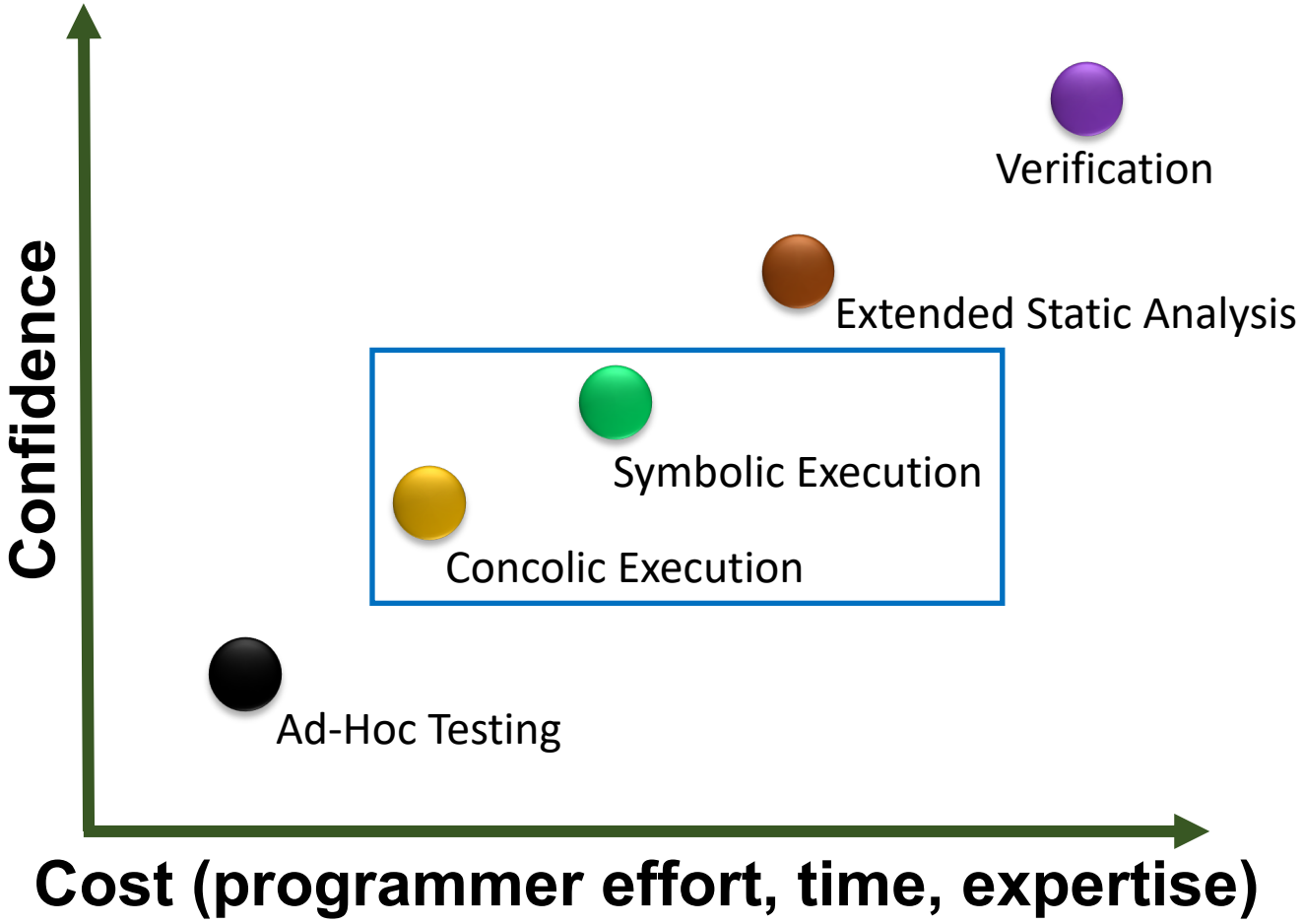
```
0. int x, y;
1. if (x>y) {
2.         x = x + y;
3.         y = x – y;
4.         x = x – y;
5.         if (x-y>0) {
6.             assert(false);
7.         }
8. }
```

- Can automatic testing prove whether the assertion fails?
  - No! Why?
- Can we automatically prove whether the assertion fails?
  - Partially Yes. How?

Testing vs. Model Checking



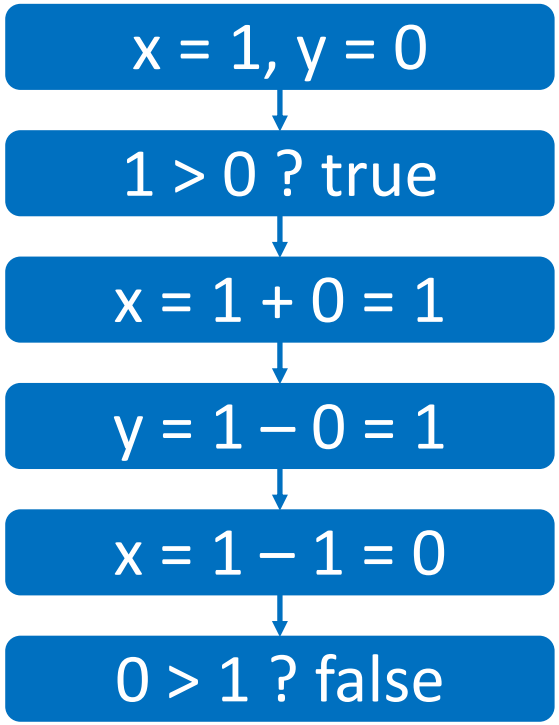
Program Validation Approaches



Symbolic Execution

Example of Standard (Concrete) Execution

```
int x, y;
if (x > y) {
  x = x + y;
  y = x – y;
  x = x – y;
  if (x > y)
    assert (false);
}
```

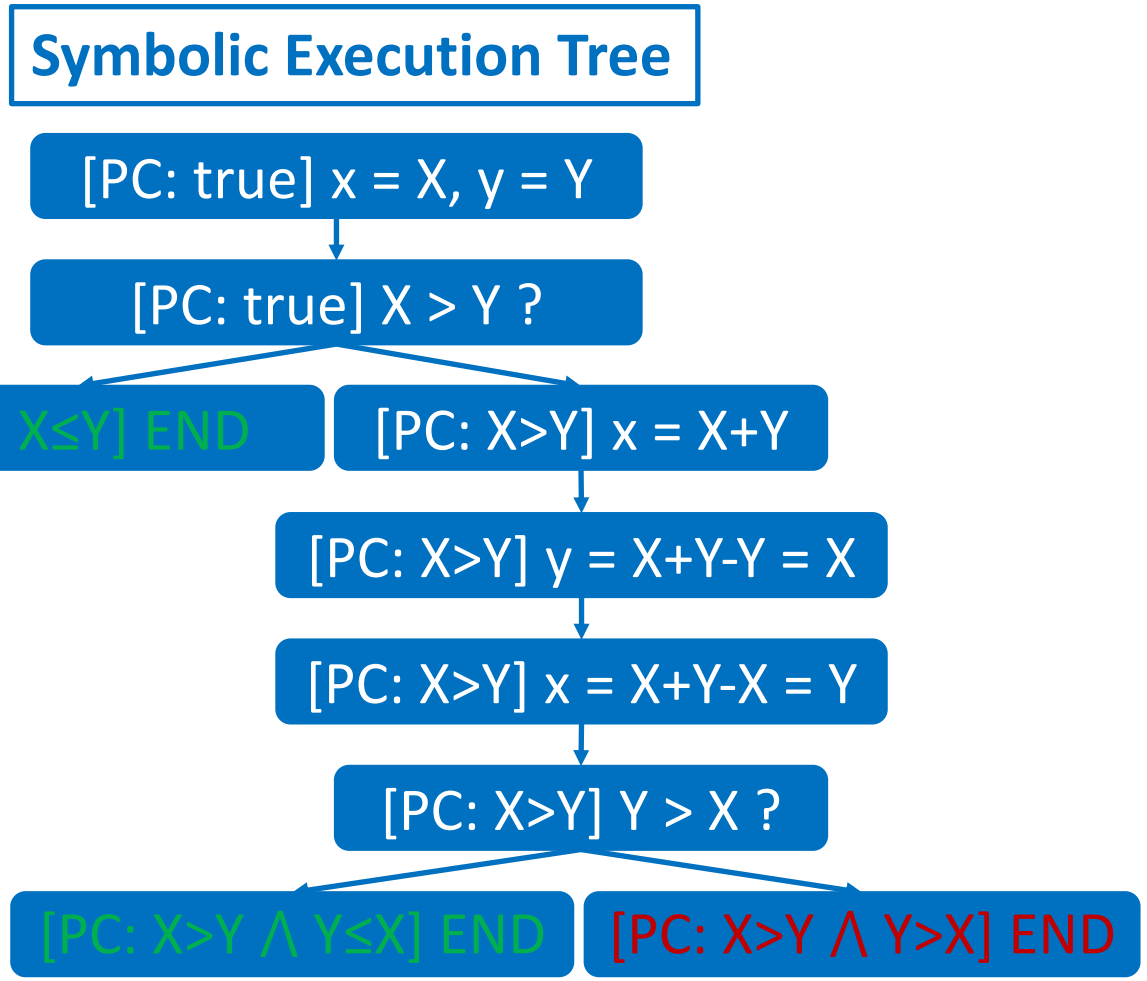


Symbolic Execution

- Analysis of programs with unspecified inputs
  - Execute a program on symbolic inputs
- Symbolic states represent sets of concrete states
- For each path, build a path condition
  - Condition on inputs – for the execution to follow that path
  - Check path condition satisfiability – explore only feasible paths
- Symbolic state
  - Symbolic expressions for variables
  - Path condition
  - Program counter
- Tools: KLEE (C), JavaPath Finder (Java), …

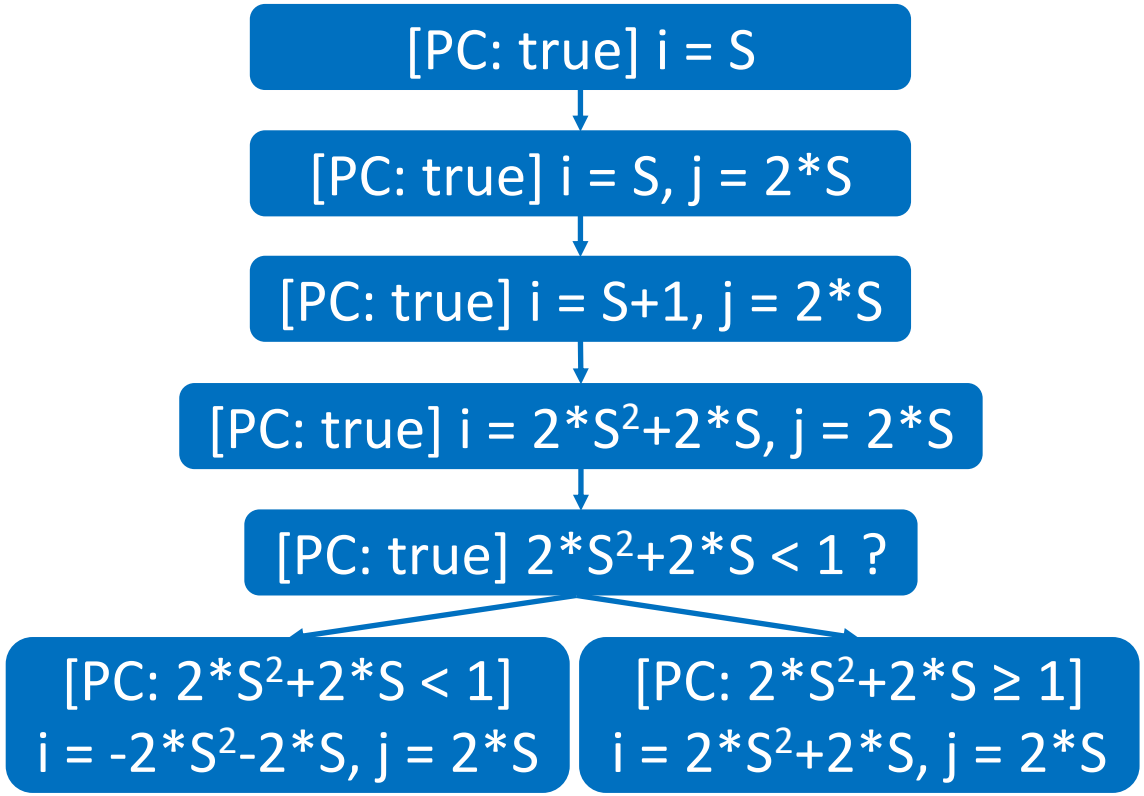
Example of Symbolic Execution

```
int x, y;
if (x > y) {
  x = x + y;
  y = x – y;
  x = x – y;
  if (x > y)
    assert (false);
}
```



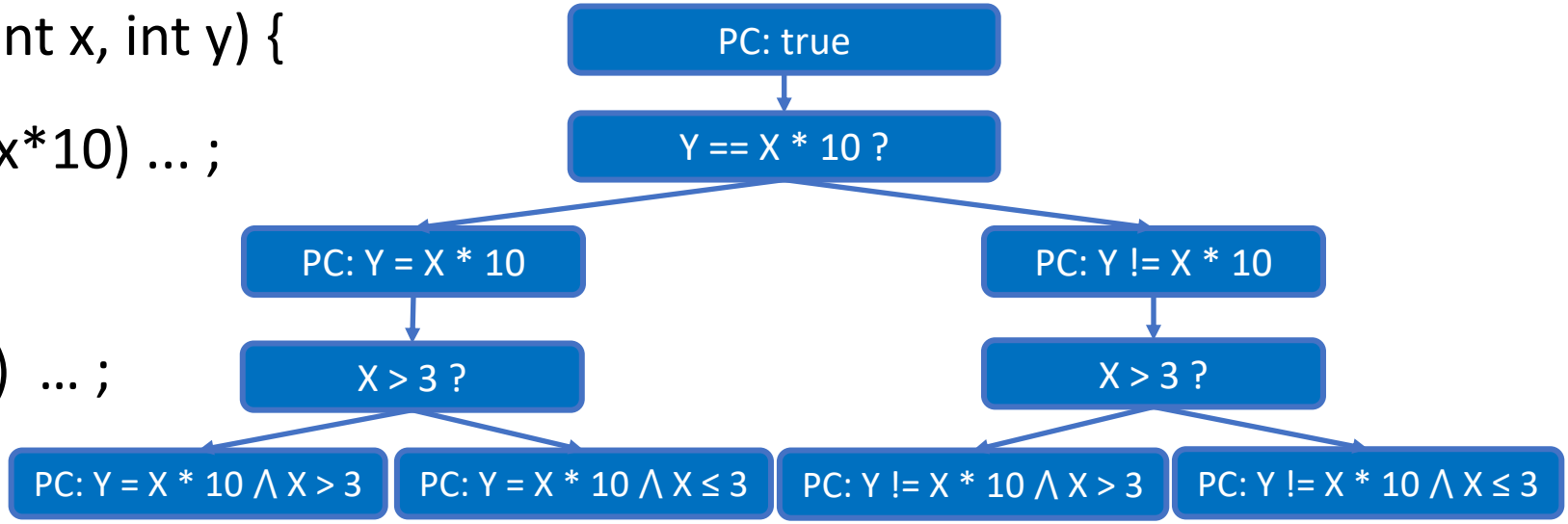
Exercise: Symbolically Execute the Program

```
int i;
int j = 2 * i;
i = i++;
i = i * j
if (i < 1)
    i = -i;
```



Exercise: Symbolically Execute the Program

```
void test(int x, int y) {
    if (y == x*10) ... ;
    else ... ;
    if (x > 3)  ... ;
    else ... ;
}
```



- How can we check the satisfiability of path conditions?
  - Constraint Solving
- How can we explore the large space of symbolic execution tree?
  - Searching Strategy

Constraint Solving

- Symbolic execution was proposed in 1976
  - L. A. Clarke, “A System to Generate Test Data and Symbolically Execute Programs”, IEEE Transactions on Software Engineering
- Become popular only in recent years due to advancement in constraint solving techniques
  - SAT Solver
  - SMT Solver
  - …

Boolean Satisfiability Problem

- **Boolean Satisfiability** (often abbreviated **SAT**) is the problem of determining if there exists an assignment that satisfies a given Boolean formula
- Consider the formula $(a \vee b) \wedge (\neg a \vee \neg c)$
  - The assignment b = true and c = false satisfies the formula
- History
  - SAT is shown to be NP-complete in 1971 (Stephen Cook)
  - Breakthrough occurred in 1990s
  - Advanced SAT solver handles problem instances with millions of Boolean variables
  - Annual competition: http://www.satcompetition.org/

SAT Solver Progress

Solvers have continually improved over time

Instance	Posit' 94
ssa2670-136	40.66s
bf1355-638	1805.21s
pret150_25	>3000s
dubois100	>3000s
aim200-2_0-no-1	>3000s
2dlx_..._bug005	>3000s
c6288	>3000s

Source: Marques-Silva 2002

Exercise: Applying SAT Solving

- Consider the following constraints
  - John can only meet either on Monday, Wednesday or Thursday; Catherine cannot meet on Wednesday; Anne cannot meet on Friday; Peter cannot meet neither on Tuesday nor on Thursday
- Question: when can the meeting take place?
  - John = (Mon ∨ Wed ∨ Thu) ∧ ¬Tue ∧ ¬Fri
  - Catherine = (Mon ∨ Tue ∨ Thu ∨ Fri) ∧ ¬Wed
  - Anne = (Mon ∨ Tue ∨ Wed ∨ Thu) ∧ ¬Fri
  - Peter = (Mon ∨ Wed ∨ Fri) ∧ ¬Tue ∧ ¬Thu
  - John ∧ Catherine ∧ Anne ∧ Peter ?
  - Monday!

SAT Extension: SMT

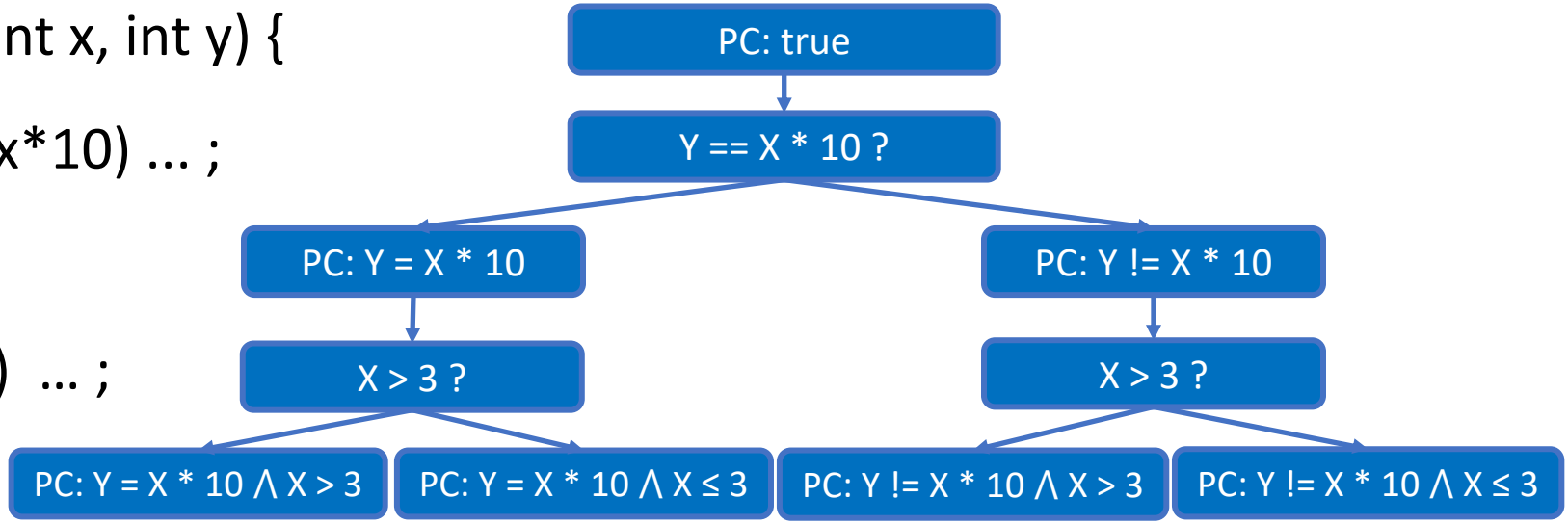
- **Satisfiability Modulo Theories** (**SMT**) enrich quantified Boolean formulas (QBF) with linear constraints, uninterpreted functions, etc.
  - QBF: Is there a solution {x, y, z} satisfying $\forall x \, \forall y \, \exists z \, (x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$?
  - Linear constraints: Is there a solution {x, y, z} satisfying $3x+2y >= 5z \wedge 5z = 2x$?
- Very efficient SMT solvers are now available that can handle many such kinds of constraints.
- Annual competition: http://www.smtcomp.org/

Optimizing SMT Queries

- Expression rewriting
  - Simple arithmetic simplifications (x * 0 --> 0)
  - Linear simplification (2 * x - x --> x)
- Constraint set simplification
  - $x < 10 \wedge x = 5$ --> $x = 5$
- Implied value concretization
  - $x + 1 = 10$ --> $x = 9$
- Constraint independence
  - $c > a \wedge b <= 5 \wedge a >= d + 10$ --> $c > a \wedge a >= d + 10, b <= 5$

Search Strategy – Breadth First Search

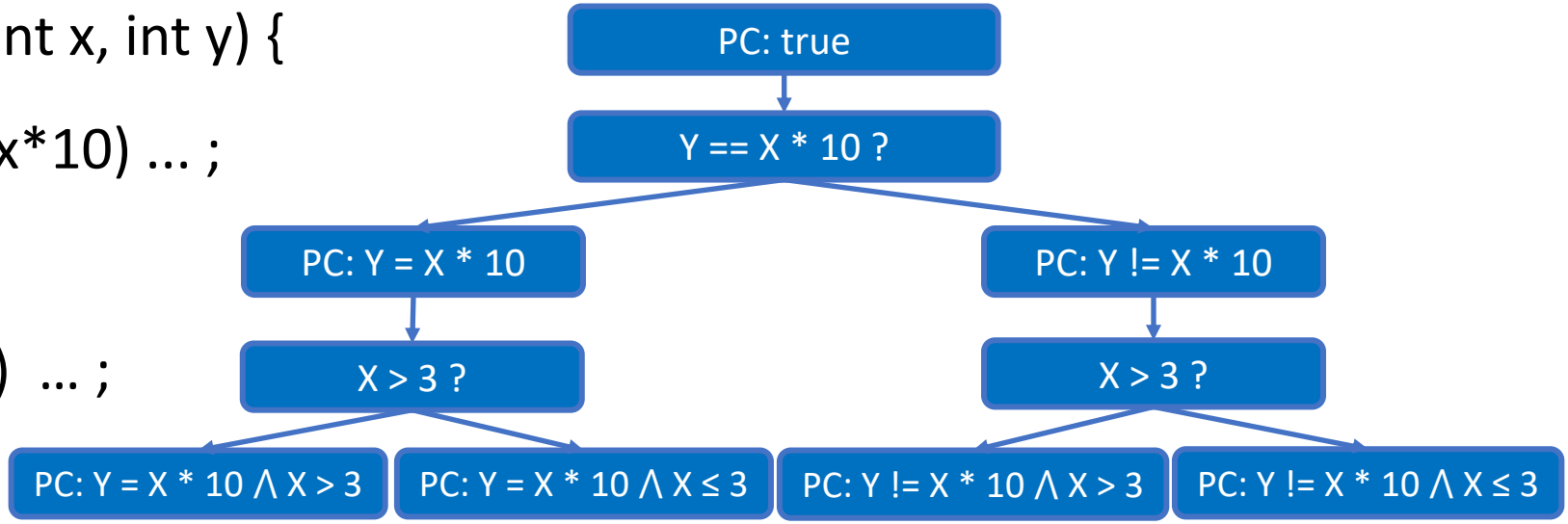
```
void test(int x, int y) {
    if (y == x*10) ... ;
    else ... ;
    if (x > 3)  ... ;
    else ... ;
}
```



Breadth first search is very slow to determine properties for a path if there are many branches

Search Strategy – Depth First Search

```
void test(int x, int y) {
    if (y == x*10) ... ;
    else ... ;
    if (x > 3)  ... ;
    else ... ;
}
```



Depth first search can stuck at somewhere in a loop

More Search Strategies

- Random Search
  - Pick next path to explore uniformly at random
  - Hard to reproduce, probably good to use pseudo-randomness based on seed
- Coverage Guided Search
  - Select paths likely to hit the new statements
  - Favor paths on recently covering new statements
  - Errors are often in hard-to-reach parts of the program, this strategy tries to reach everywhere
- Generational Search
  - Generation 0: pick one path at random, run to completion
  - Generation 1: take the path from generation 0, negate one branch condition on the path to yield a new path prefix, find a solution for that path prefix, and then take the resulting path
  - …

Applications of Symbolic Execution

- Generating test inputs
- Detecting infeasible paths
- Finding bugs and vulnerabilities
  - E.g., divide by zero bugs
- Proving two code segments are equivalent
- Generating program invariants
- Debugging
- Repairing programs
- …

Challenges – Path Explosion

- Exponential in branch structures

```
1. int a = input(), b = input();
2. Int c = input(), d = input();
2. if (a) {…} else {…}
3. if (b) {…} else {…}
4. if (c) {…} else {…}
4. if (d) {…} else {…}
```

4 variables and $2^4 = 16$ program paths!

- Even worse, loops on symbolic variables

```
1. int x = input();
2. while (x > 0) {
3.       x++;
4.       assert(…);
5. }
```

Potentially 2^{31} paths through the loop!

Challenges – Constraint Solving

- The state-of-the-art: support theories on linear integer arithmetic, bit vectors, string, etc., but have limited support for complex constraints and are not particularly scalable
  - Non-linear constraints, e.g., sin(), cos()
  - Float-point constraints, e.g., 2.5 * x – 2.9 * y > 1.1
  - …

```
1. int x = input(), y = input(), z = input();
2. if (5x^63 + 7x^12 = 78y^2 + z) {
3.       assert(false);
4. }
```

Challenges – Complex Code and Environment Dependencies

- Complex code
  - Data structures and pointers
- Environment dependencies
  - Native calls
  - System calls
  - Library calls
  - File operations
  - Network events
  - …

Potential Solutions

- Path explosion
  - Heuristic-based search strategies
  - Limit the exploration depth (but lead to incompleteness)
  - …
- Constraint solving
  - Meta-heuristic-based solvers (but lead to imprecise)
  - …
- Complex code and environment dependencies
  - Build simple versions of library calls
  - Simulate system calls
  - Concretization
  - …

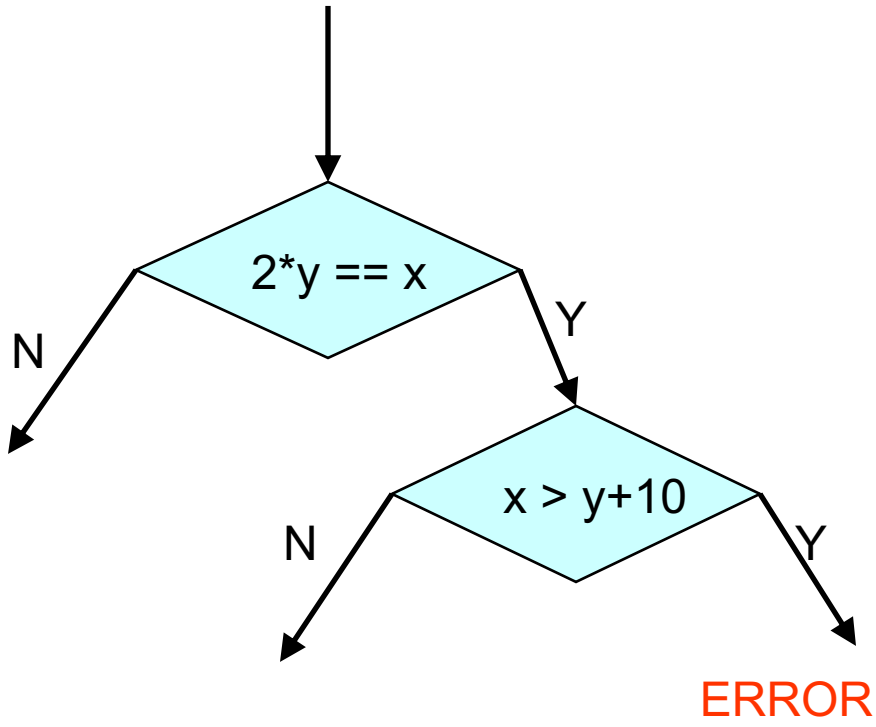
Concolic Execution

Concolic (Concrete + Symbolic) Execution

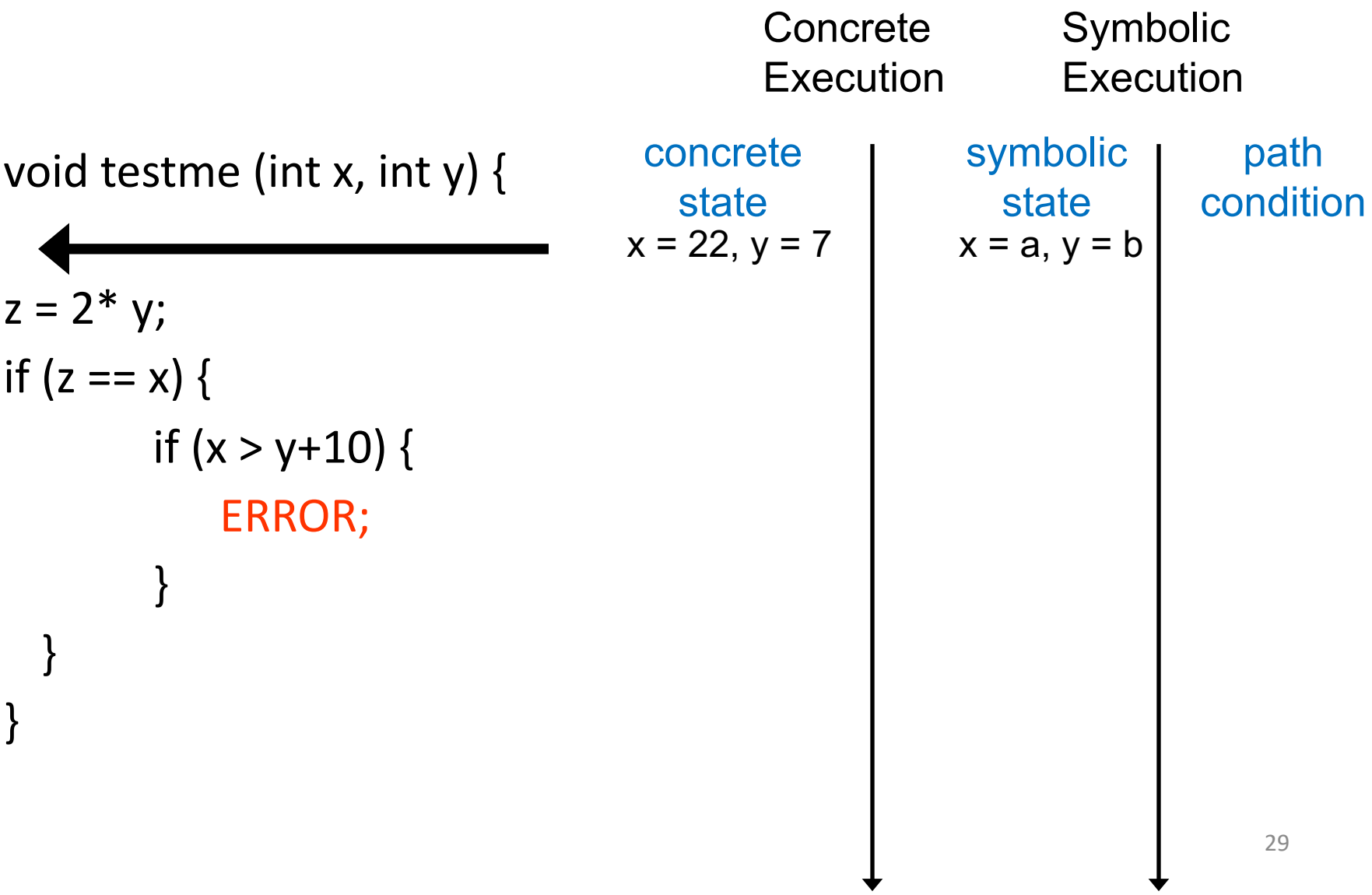
- Execute program with both concrete and symbolic inputs (also known as dynamic symbolic execution)
- The intention is to visit deep into the program execution tree
- Steps
  - Generate a random test input
  - Execute the program with the random test input and the symbolic input to collect the path constraint
  - Negate the last conjunct to obtain a constraint
  - Solve it to get test input to another path and execute the input

Example

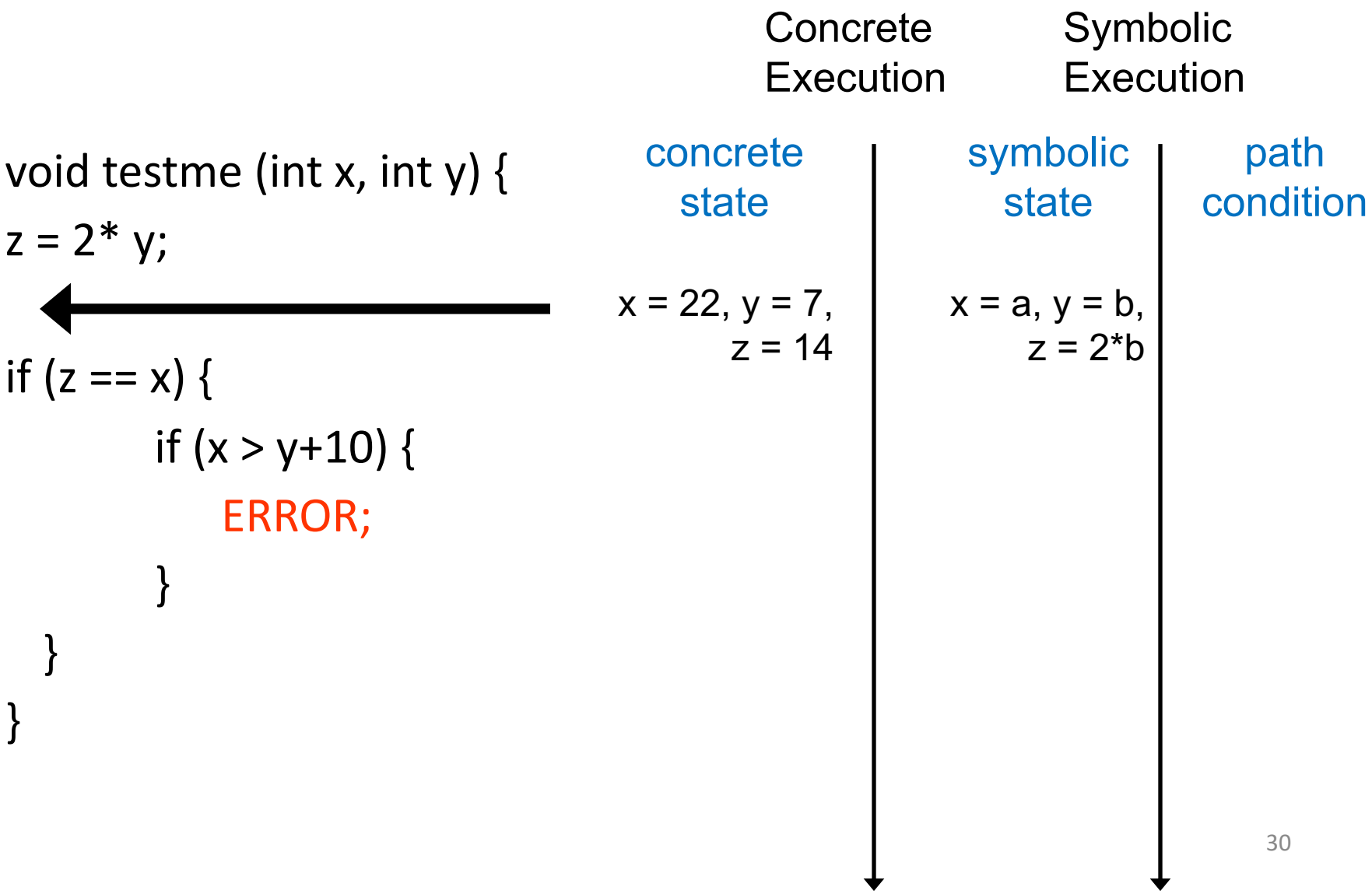
```
void testme (int x, int y)
{
      z = 2*y;
      if (z == x) {
            if (x > y+10) {
                  ERROR;
            }
      }
}
```



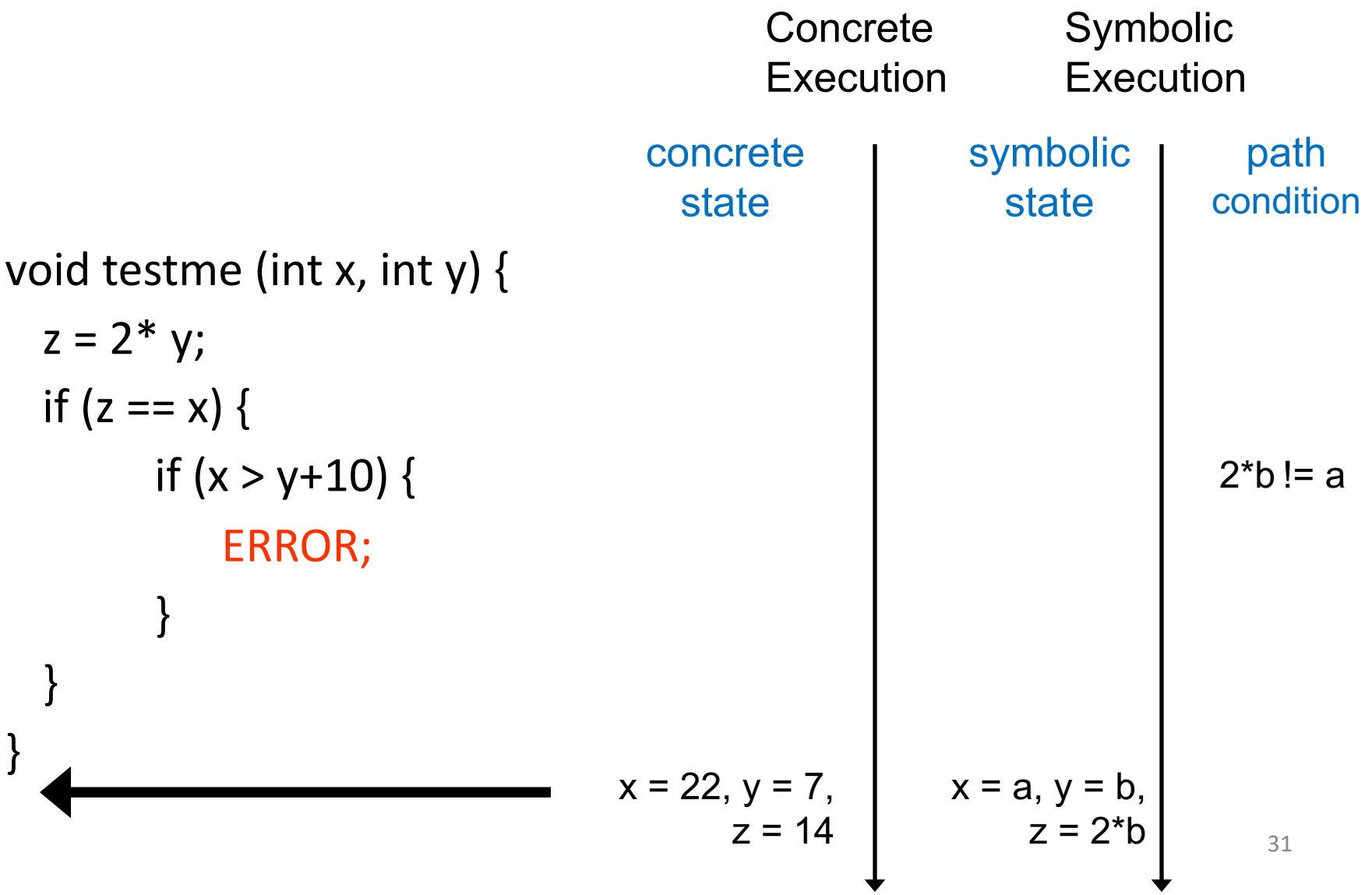
Example of Concolic Execution



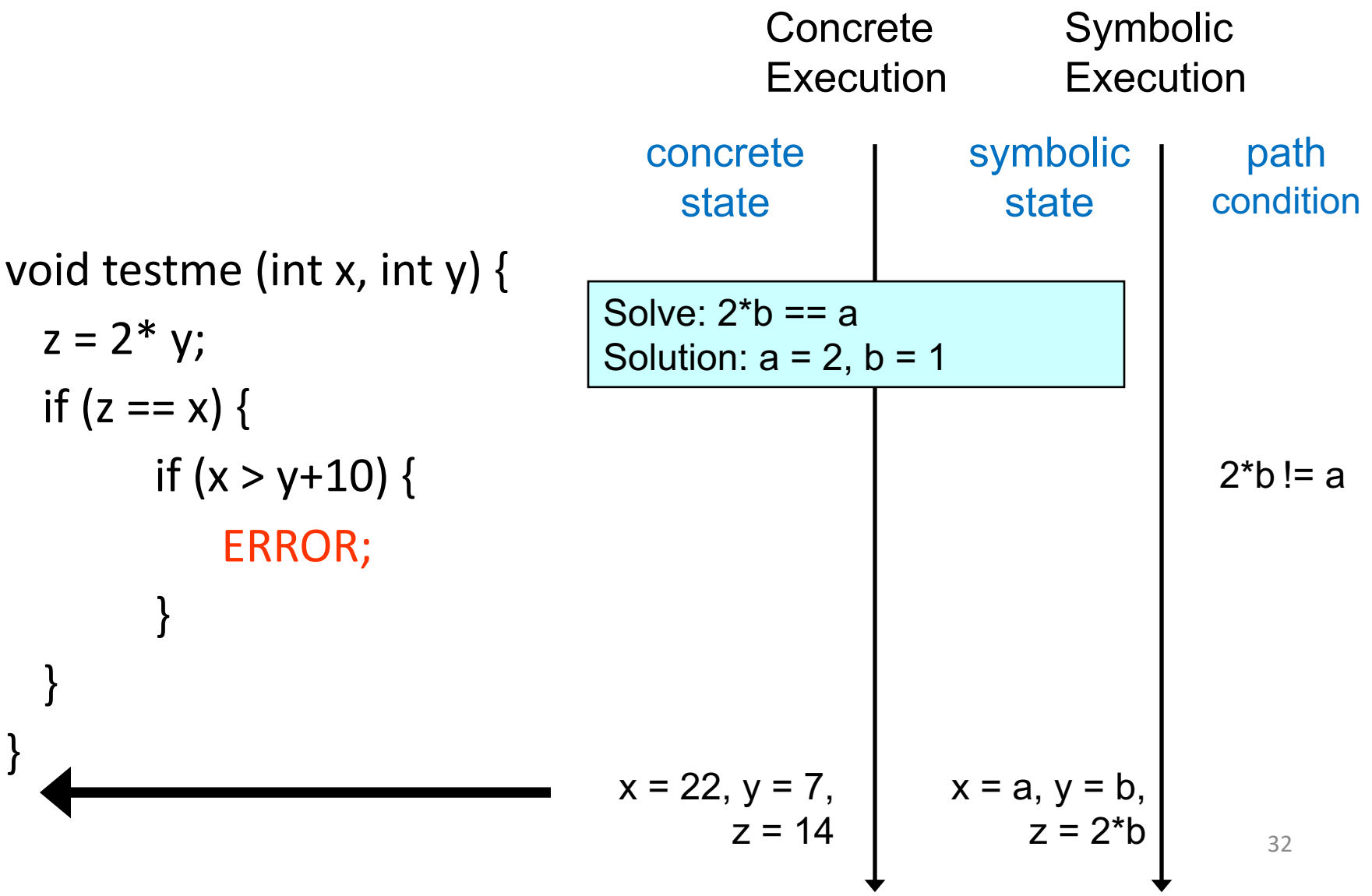
Example of Concolic Execution



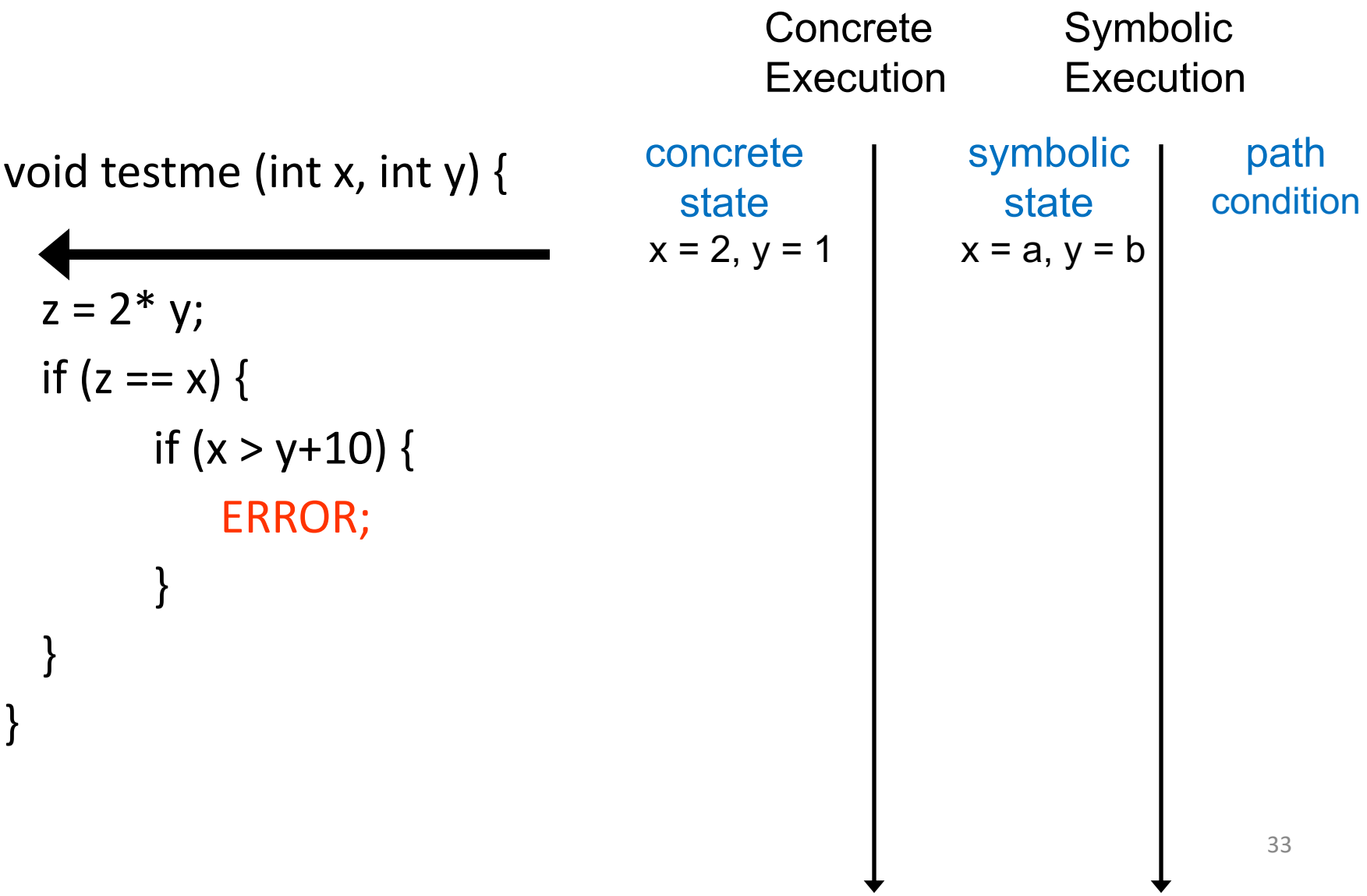
Example of Concolic Execution



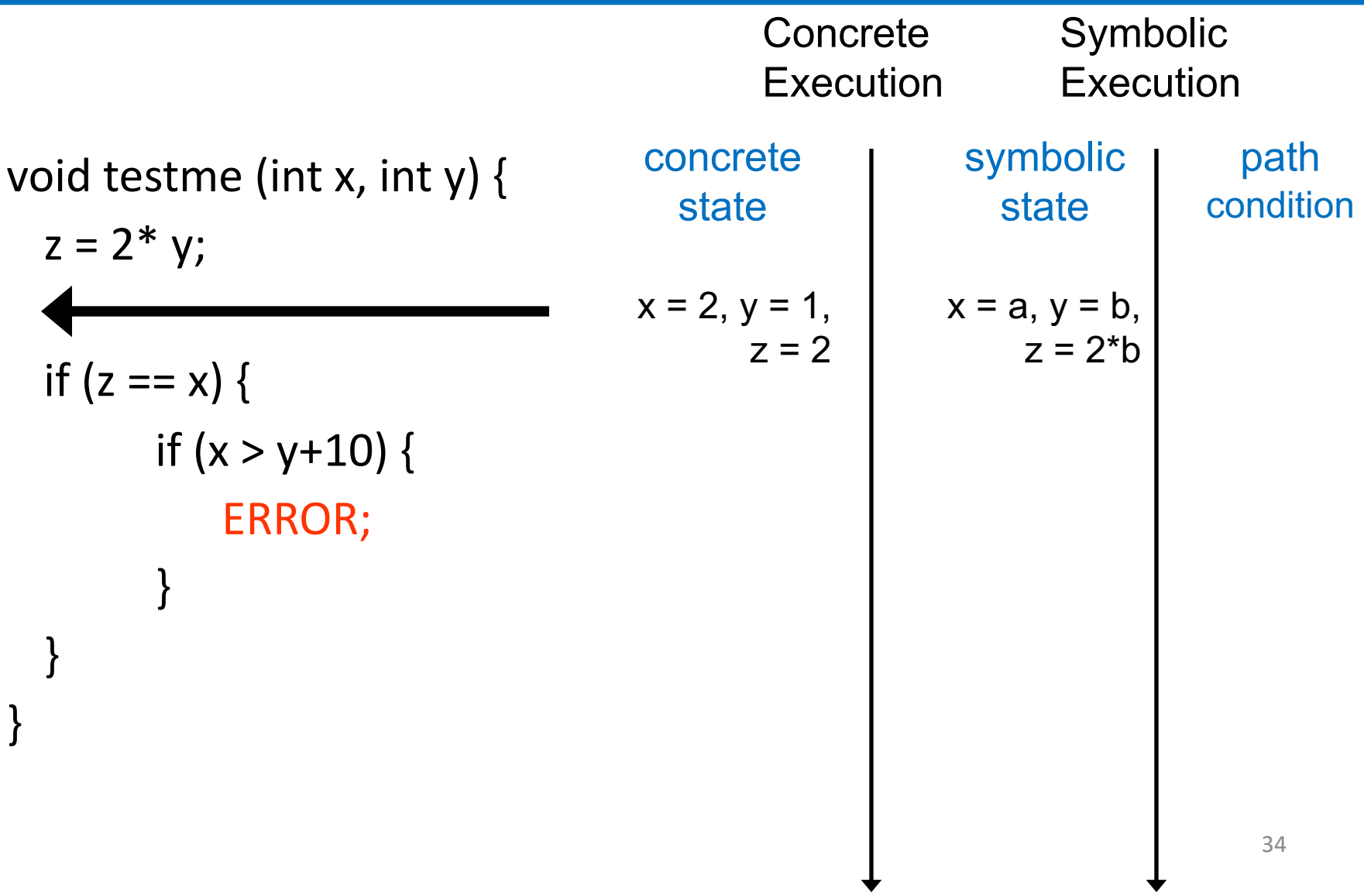
Example of Concolic Execution



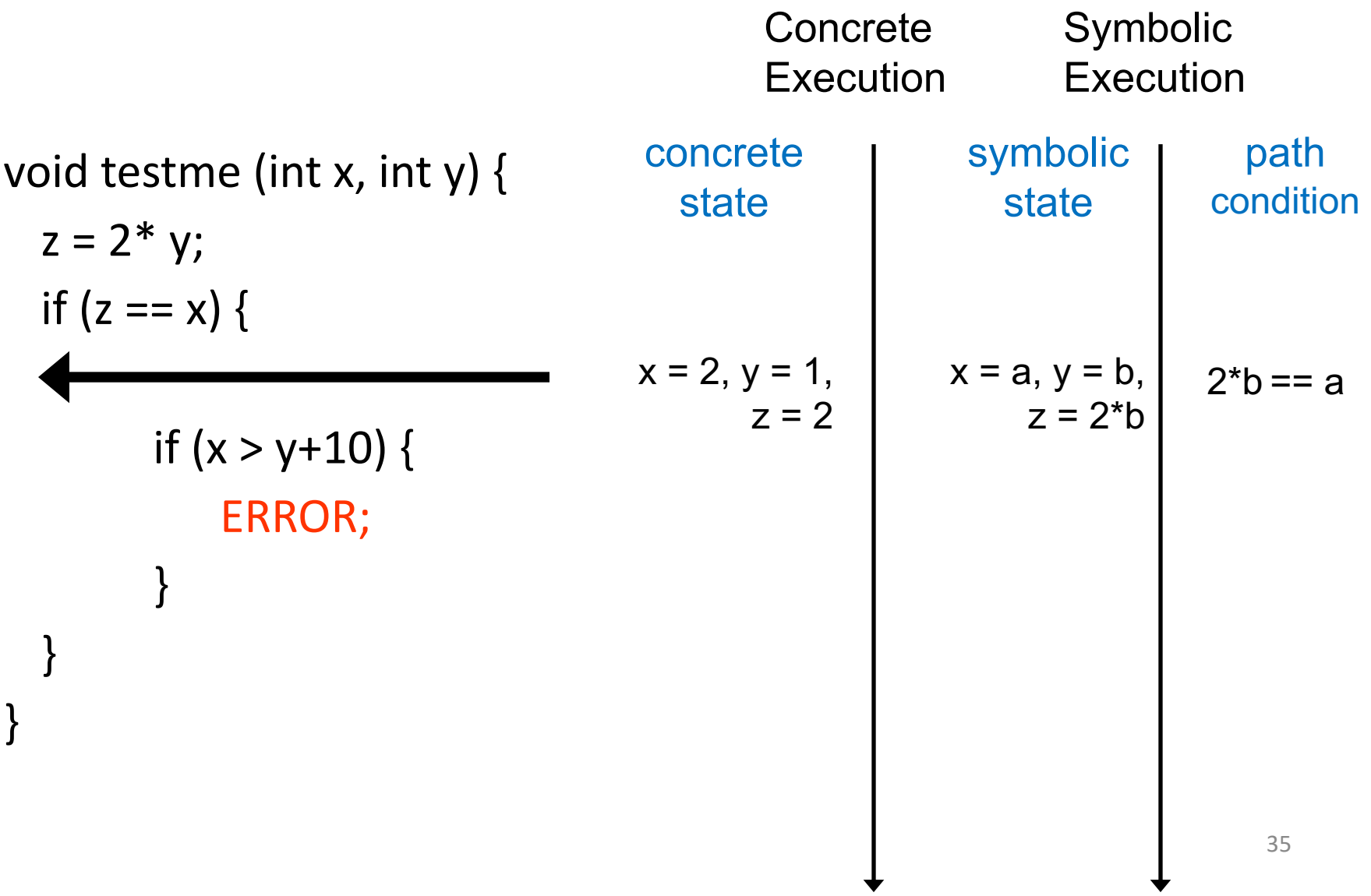
Example of Concolic Execution



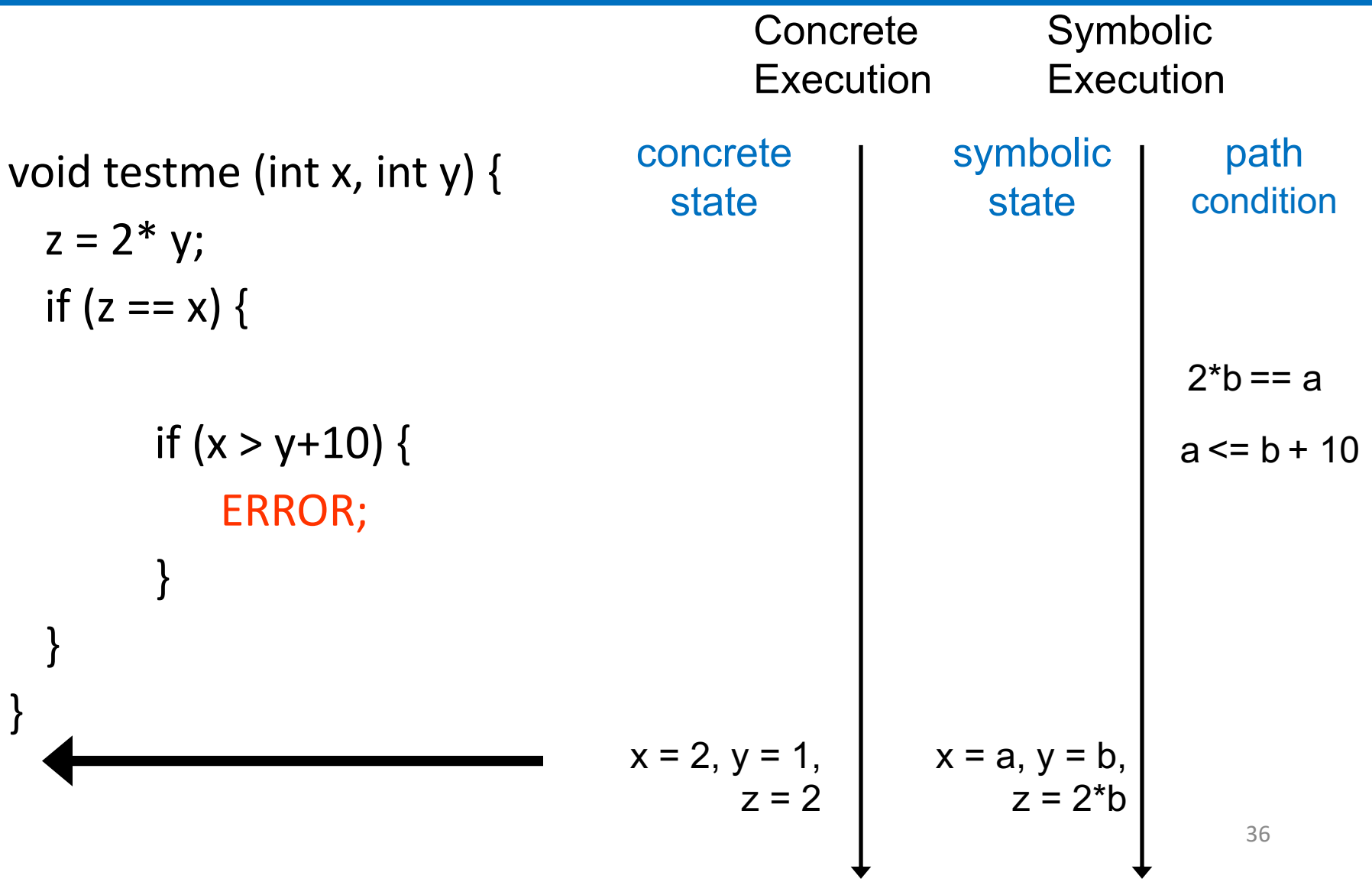
Example of Concolic Execution



Example of Concolic Execution

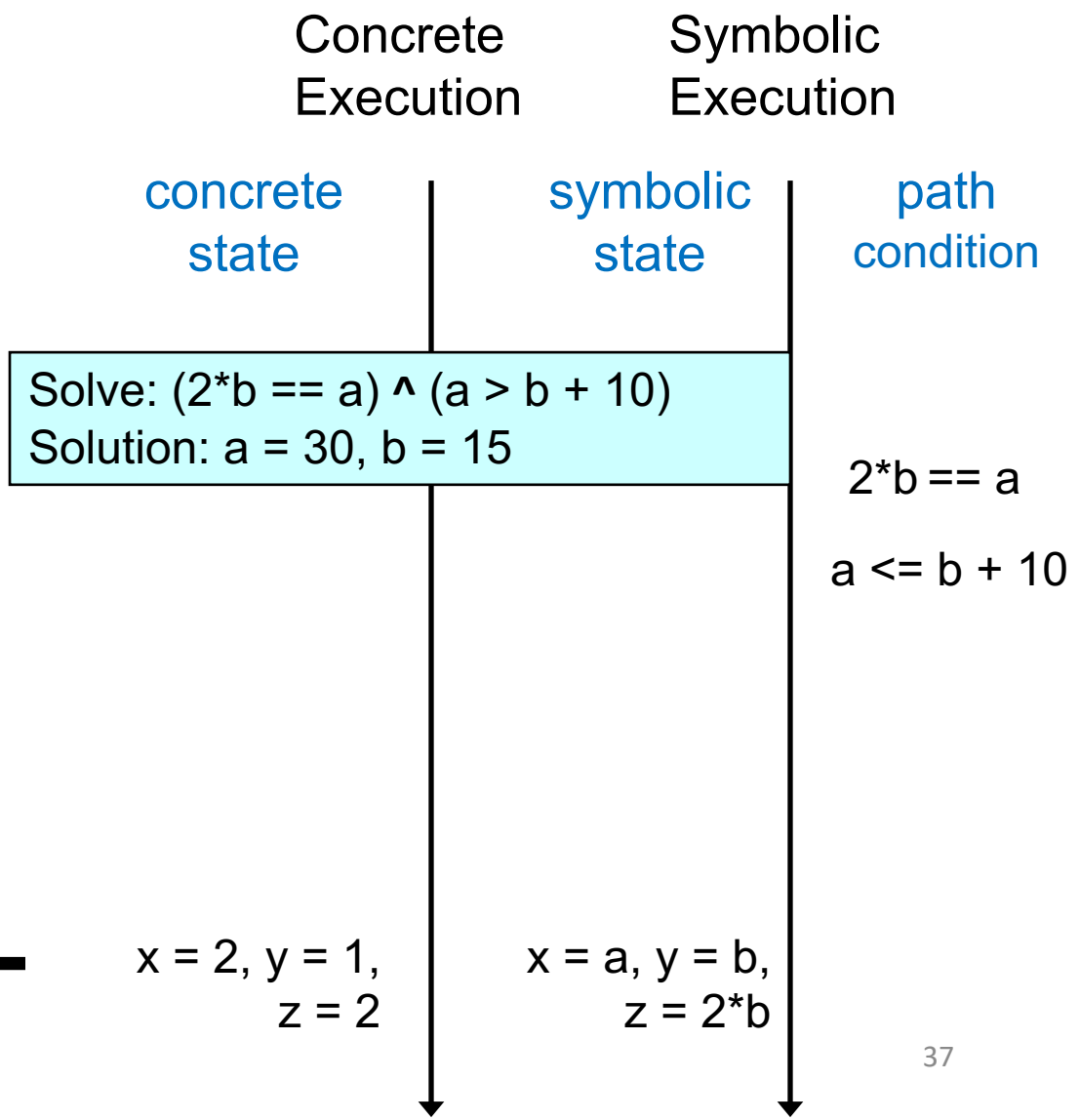


Example of Concolic Execution

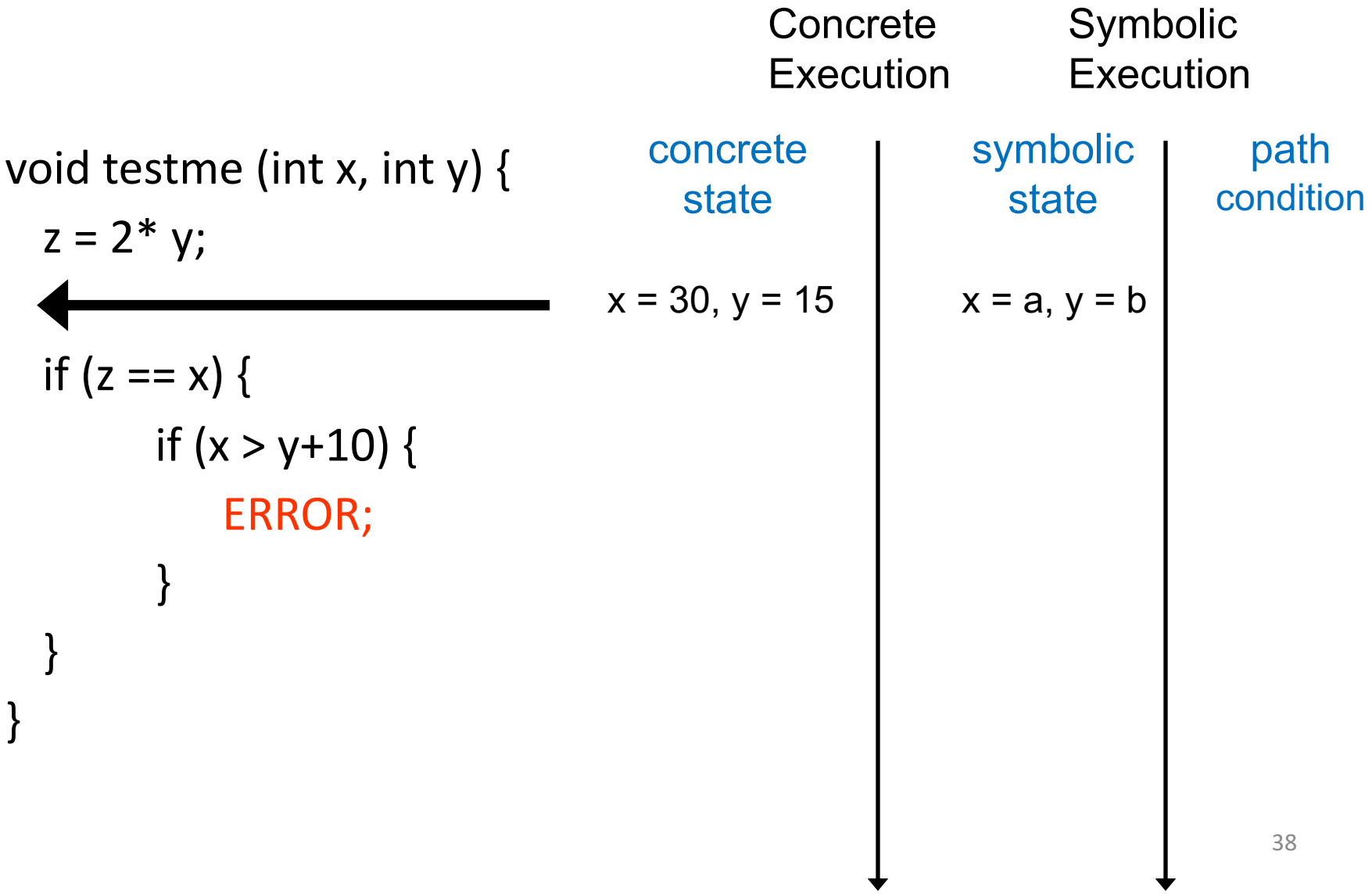


Example of Concolic Execution

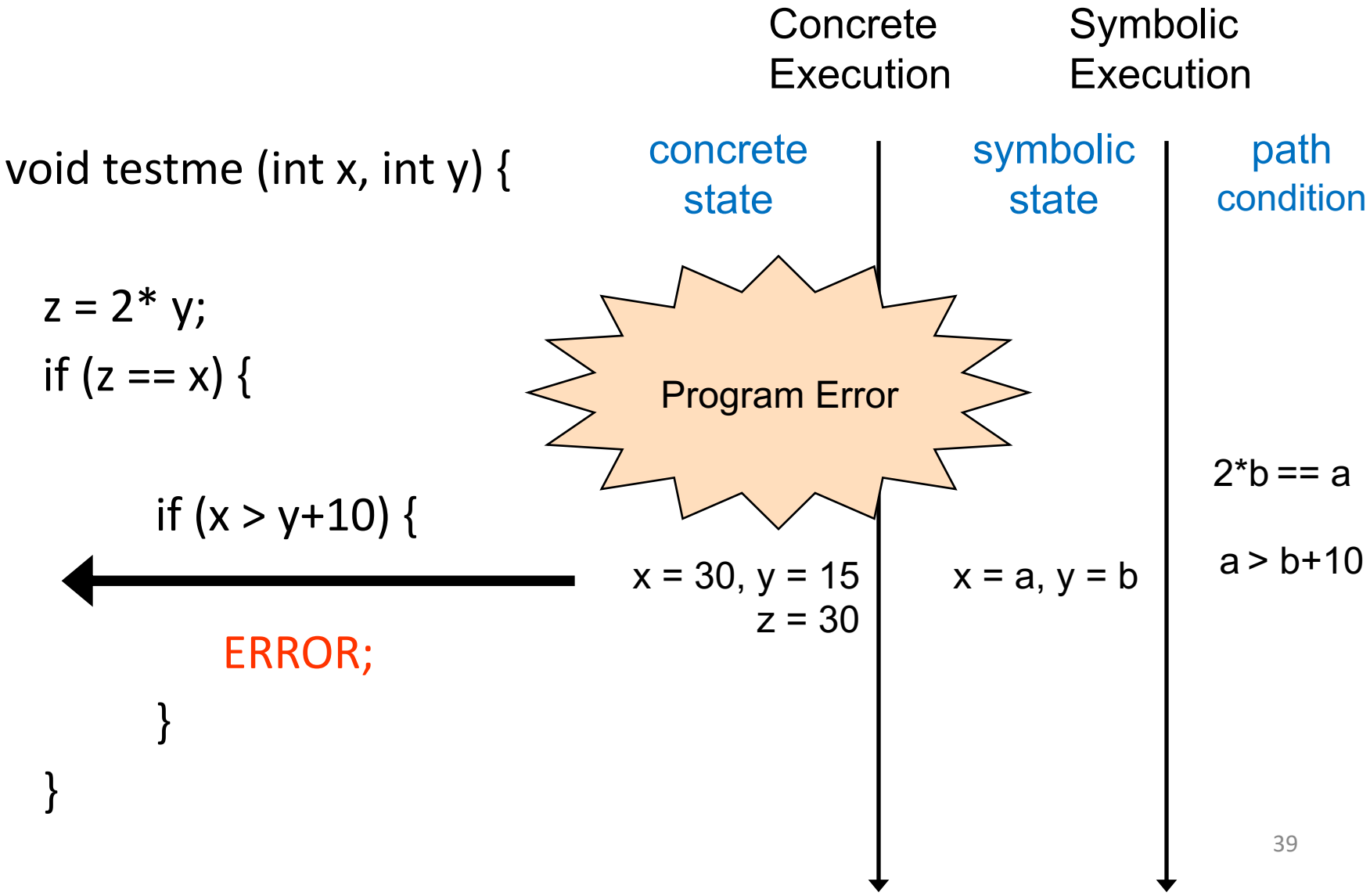
```
void testme (int x, int y) {
   z = 2* y;
   if (z == x) {

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



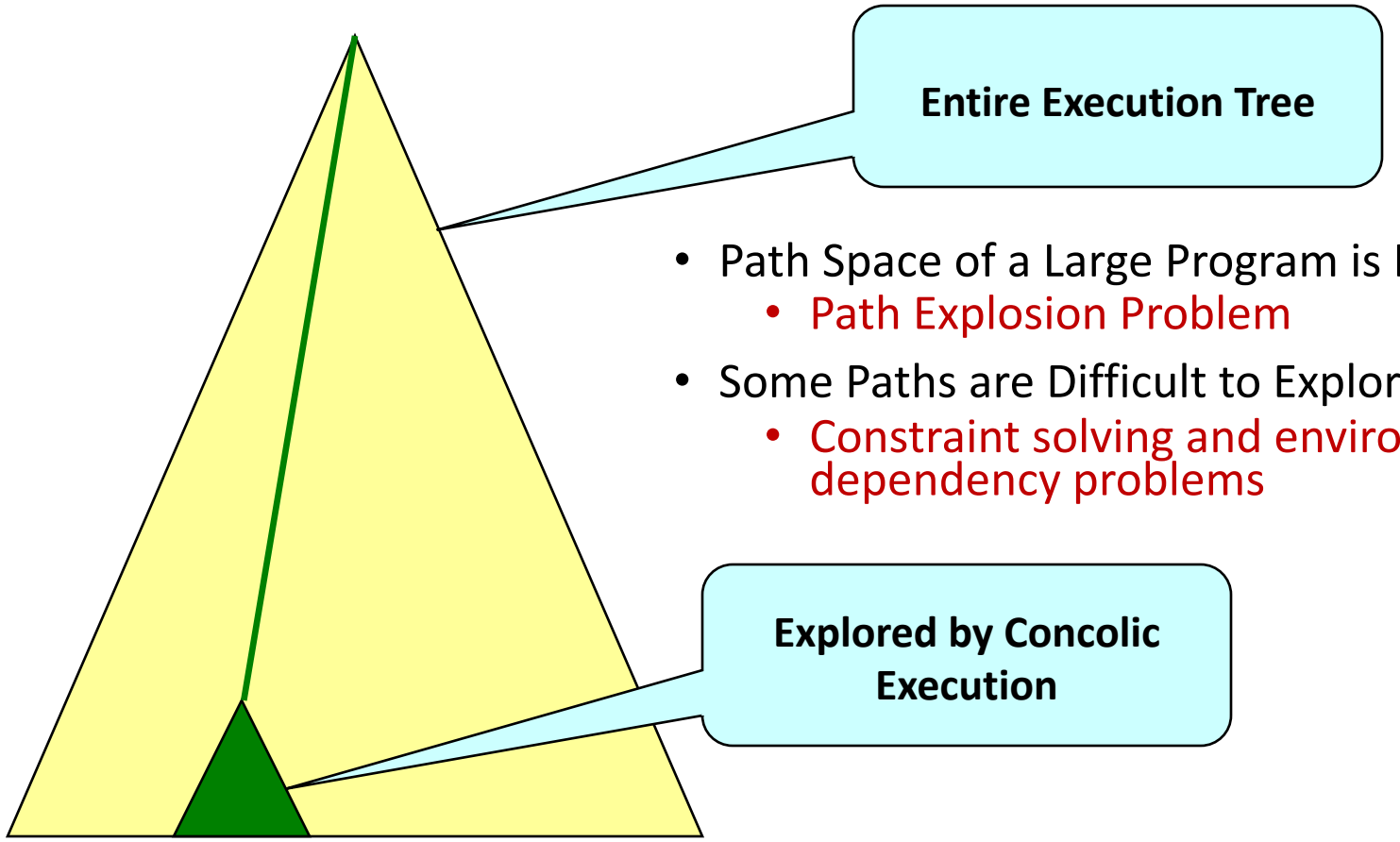
Example of Concolic Execution



Example of Concolic Execution



Bird's View



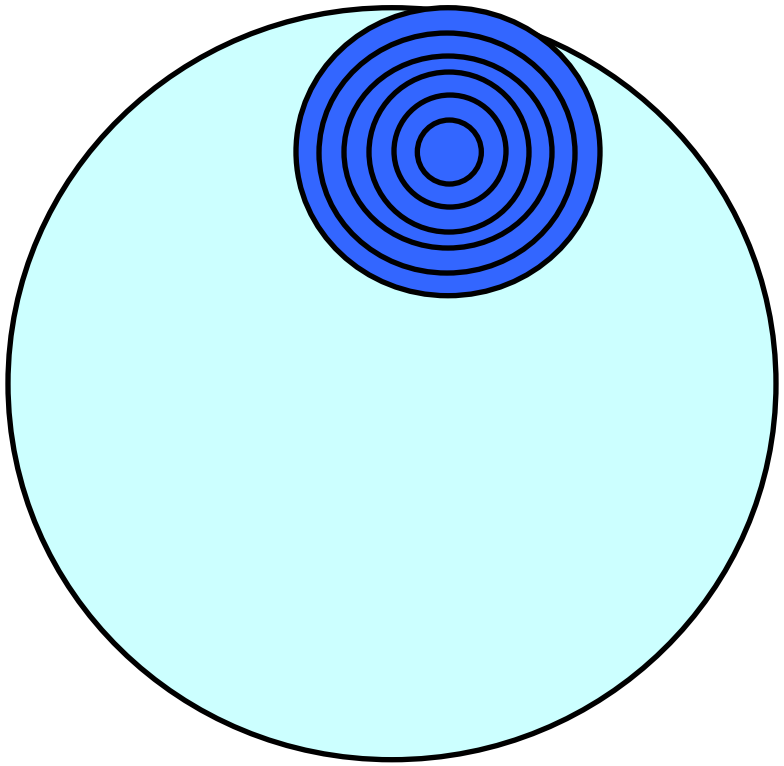
- Path Space of a Large Program is Huge
  - Path Explosion Problem
- Some Paths are Difficult to Explore
  - Constraint solving and environment dependency problems

An Example

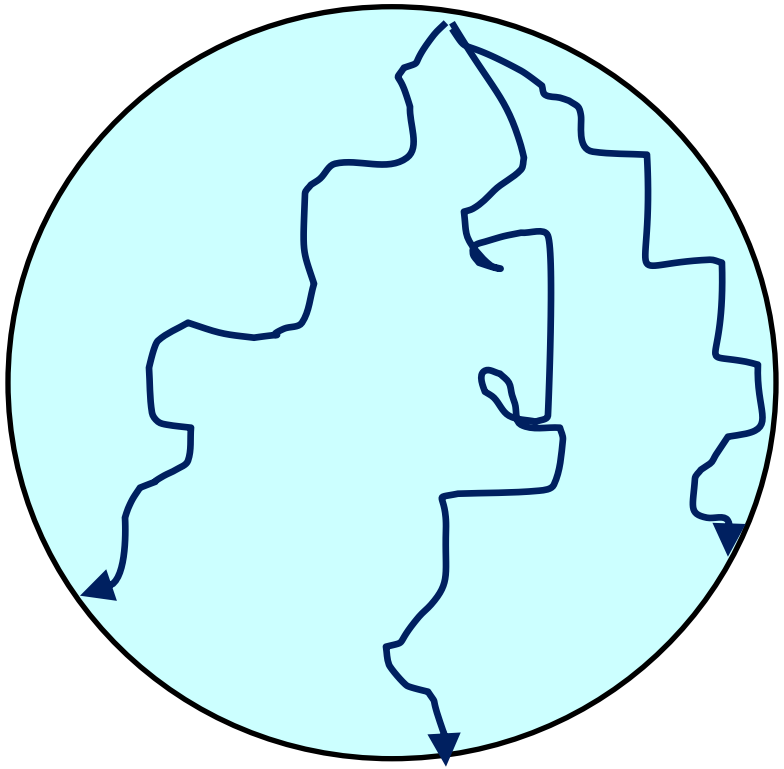
```
1.  foobar(int x, int y){
2.    if (x*x*x > 0){
3.      if (x>0 && y==10){
4.        abort();
5.      }
6.    } else {
7.      if (x>0 && y==20){
8.        abort();
9.      }
10.   }
11. }
```

- Random testing gets stuck at line 3
- Symbolic execution gets stuck at line 2
- Concolic execution finds the only error at line 4
  - Use concrete value for x

Concolic Execution vs. Random Testing



Concolic: Broad, Shallow



Random: Narrow, Deep

An Example

```
Example ( ) {
1: state = 0;
2: while(1) {
3:   s = input();
4:   c = input();
5:   if(c==':' && state==0)
       state=1;
6:   else if(c=='\n' && state==1)
       state=2;
7:   else if (s[0]=='I' &&
       s[1]=='C' &&
       s[2]=='S' &&
       s[3]=='E' &&
       state==2) {
         COVER_ME;
    }
  }
}
```

COVER_ME can be hit on an input

- s = "ICSE"
- c = ':' '\n'

Similar code in

- Text editors (vi)
- Parsers (lexer)
- Event-driven programs (GUI)

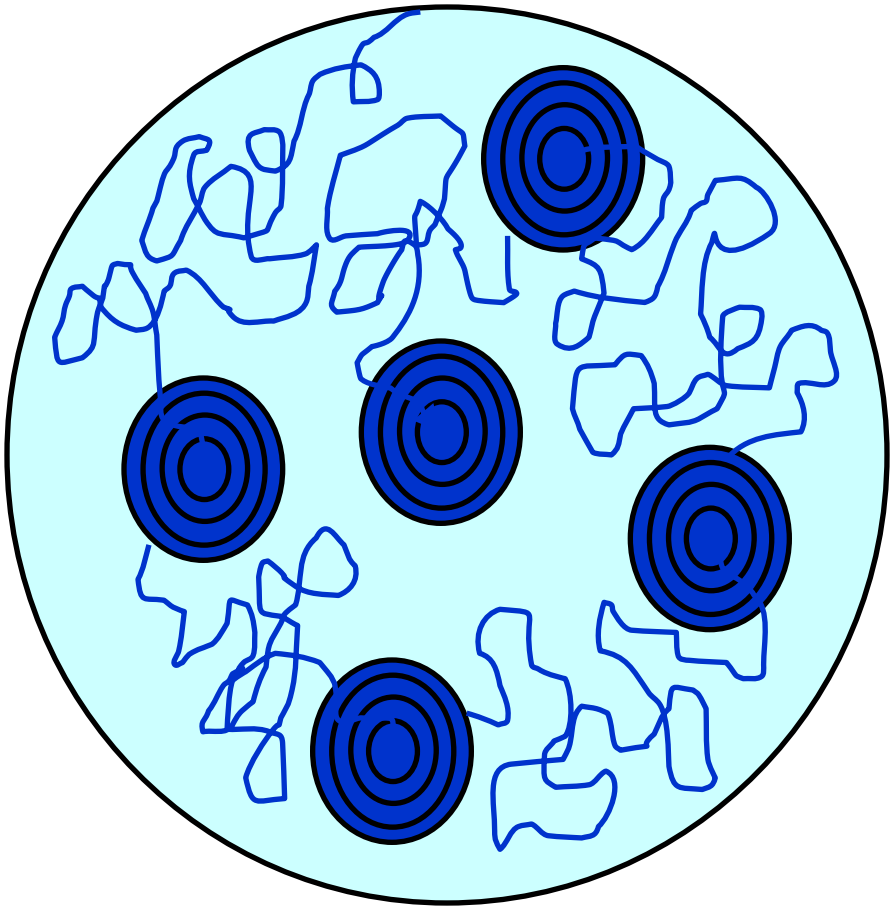
An Example

```
Example ( ) {
1: state = 0;
2: while(1) {
3:   s = input();
4:   c = input();
5:   if(c==':' && state==0)
       state=1;
6:   else if(c=='\n' && state==1)
       state=2;
7:   else if (s[0]=='I' &&
       s[1]=='C' &&
       s[2]=='S' &&
       s[3]=='E' &&
       state==2) {
         COVER_ME;
    }
  }
}
```

- **Pure random testing** can get to state = 2, but difficult to get 'ICSE'
- Probability $1/(2^8)^6 = 3*10^{-15}$

- **Concolic testing** can generate 'ICSE' but explores many paths to get to state = 2

Hybrid Concolic Testing



```
while (not required coverage) {
    while (not saturation)
        perform random testing;
    Checkpoint;
    while (not increase in coverage)
        perform concolic testing;
    Restore;
}
```

Interleave random testing and concolic testing to increase coverage

Deep and broad search

Hybrid Concolic Testing

```
Example ( ) {
1: state = 0;
2: while(1) {
3:   s = input();
4:   c = input();
5:   if(c==':' && state==0)
       state=1;
6:   else if(c=='\n' && state==1)
       state=2;
7:   else if (s[0]=='I' &&
       s[1]=='C' &&
       s[2]=='S' &&
       s[3]=='E' &&
       state==2) {
         COVER_ME;
    }
  }
}
```

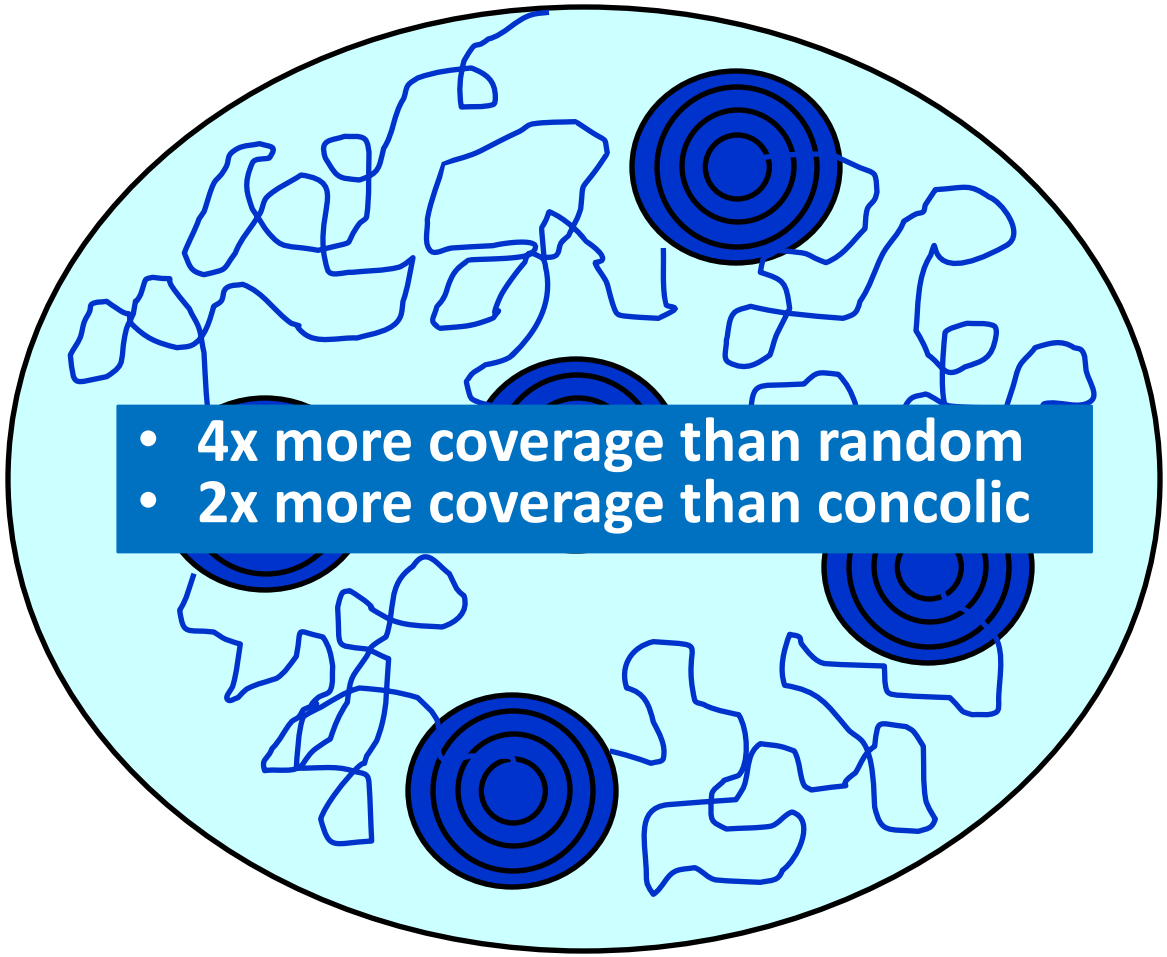
Random Testing Phase

- '$', '&', '-', '6', ':', '%', '^', '\n', 'x', '~' …
- Saturate after many (~10000) iterations
- In less than 1 second
- **COVER_ME** is not reached

Concolic Testing Phase

- s[0]='I', s[1]='C', s[2]='S', s[3]='E'
- Reach **COVER_ME**

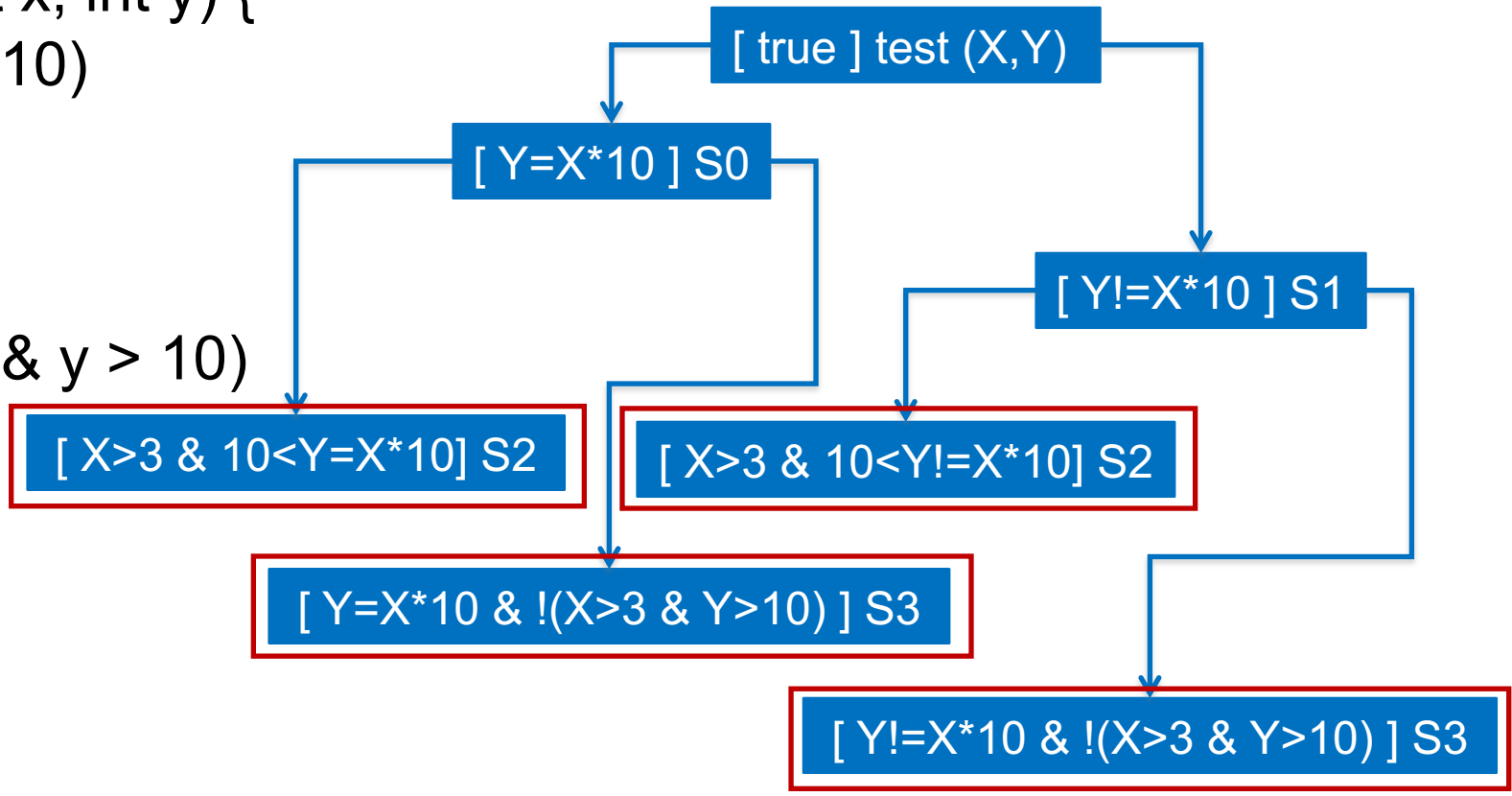
Hybrid Concolic Testing



Probabilistic Symbolic Execution

Symbolic Execution

```
void test(int x, int y) {
   if (y == x*10)
     S0;
   else
     S1;
   if (x > 3 && y > 10)
     S2;
   else
     S3;
}
```



What is the probability of reaching these state?

Probabilistic Symbolic Execution

>=

$$PC = C_1 \,\&\, C_2 \,\&\, \ldots \,\&\, C_n$$

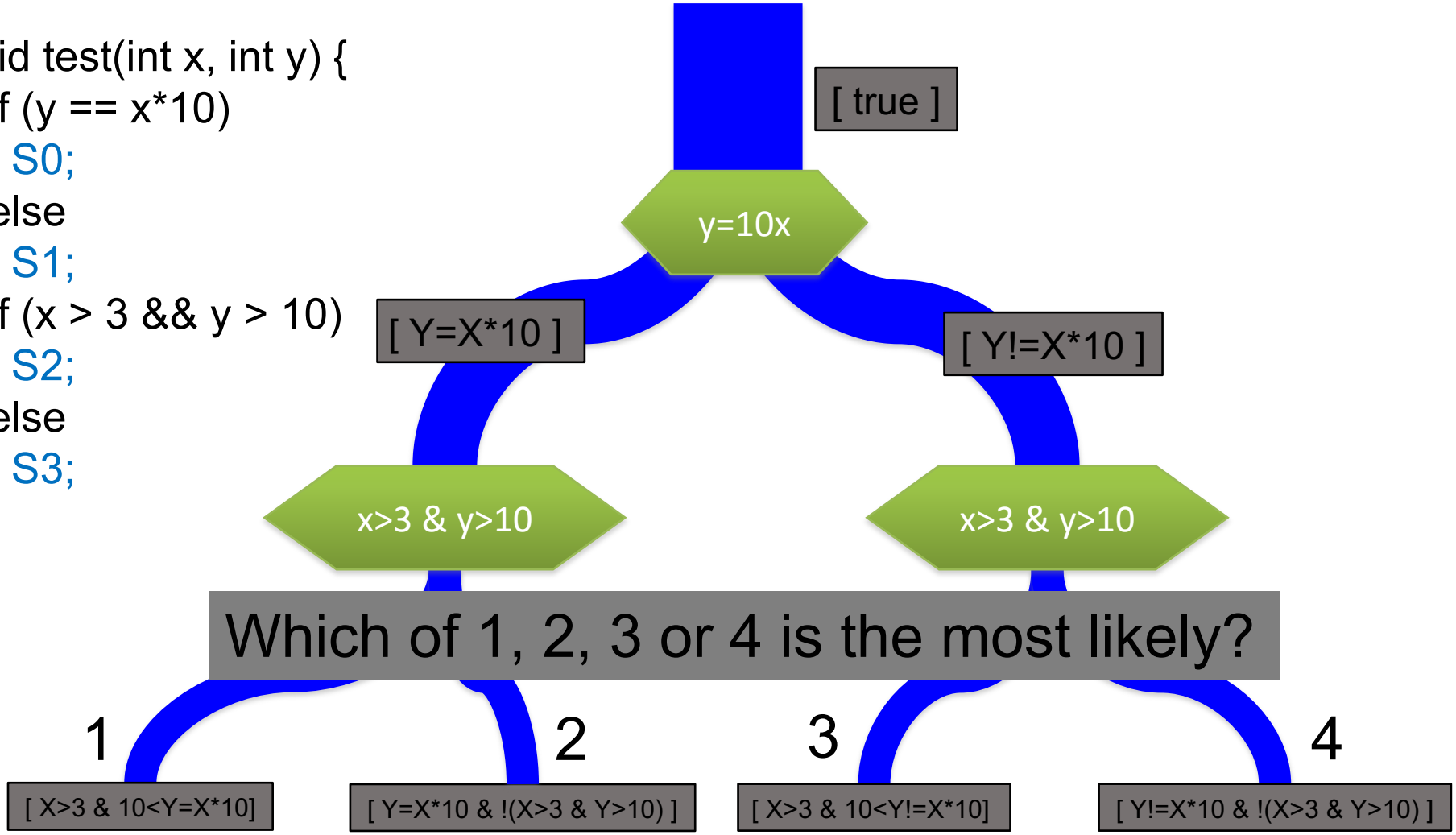
PC solutions

PC feasibility

>0

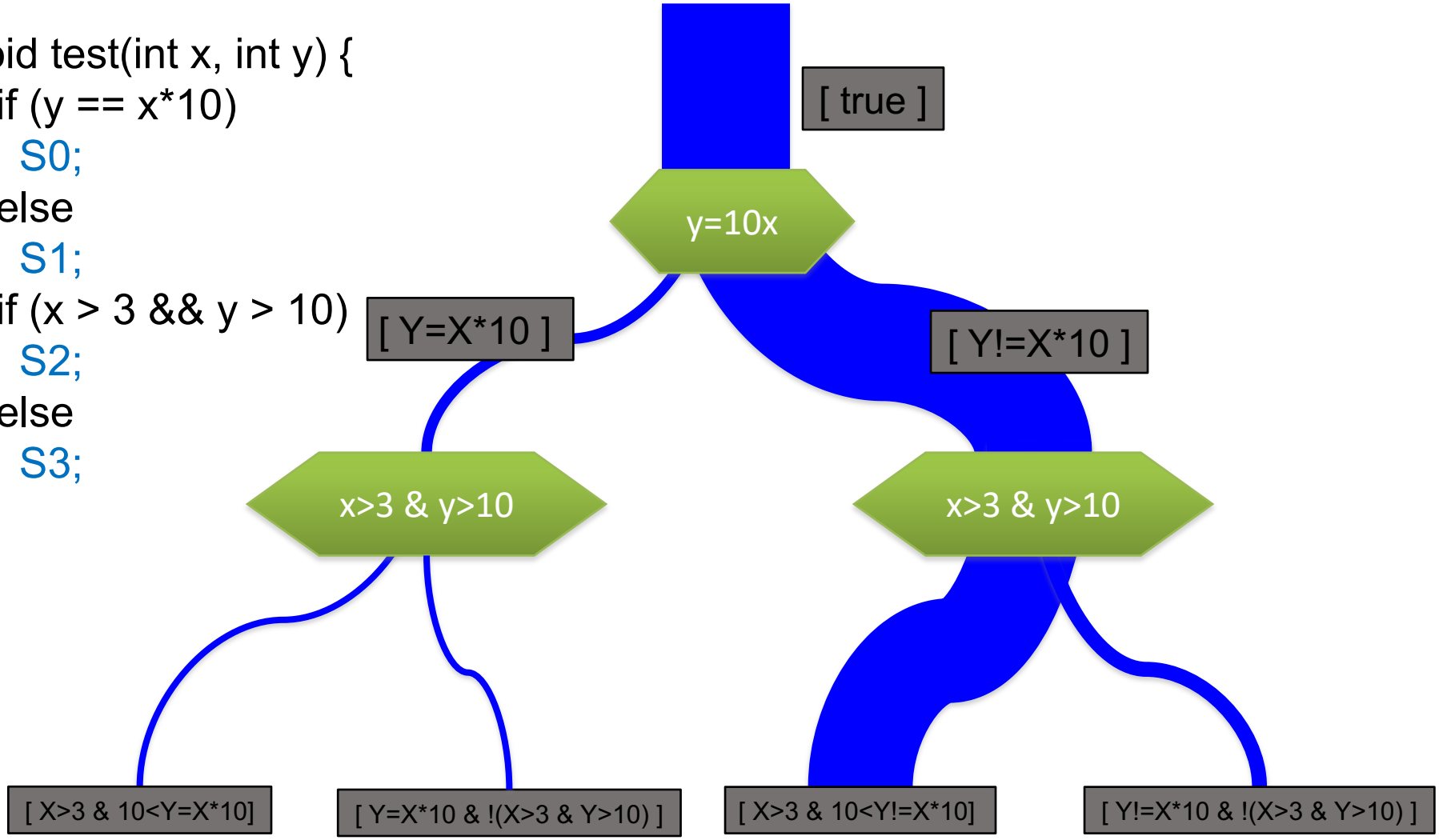
Paths and Rivers

```
void test(int x, int y) {
   if (y == x*10)
     S0;
   else
     S1;
   if (x > 3 && y > 10)
     S2;
   else
     S3;
}
```



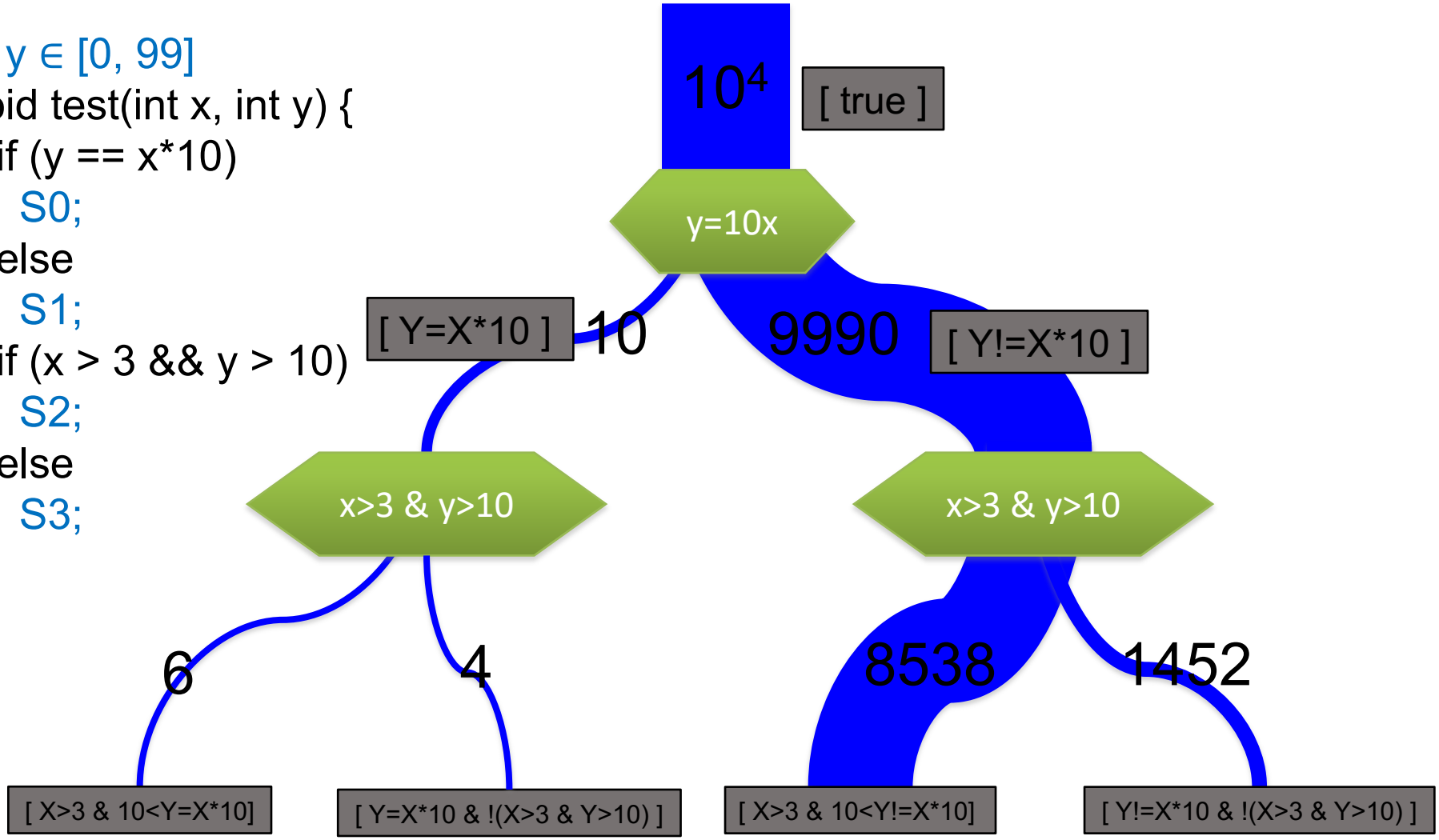
Paths and Rivers

```
void test(int x, int y) {
   if (y == x*10)
     S0;
   else
     S1;
   if (x > 3 && y > 10)
     S2;
   else
     S3;
}
```



Model Counting (e.g., LattE)

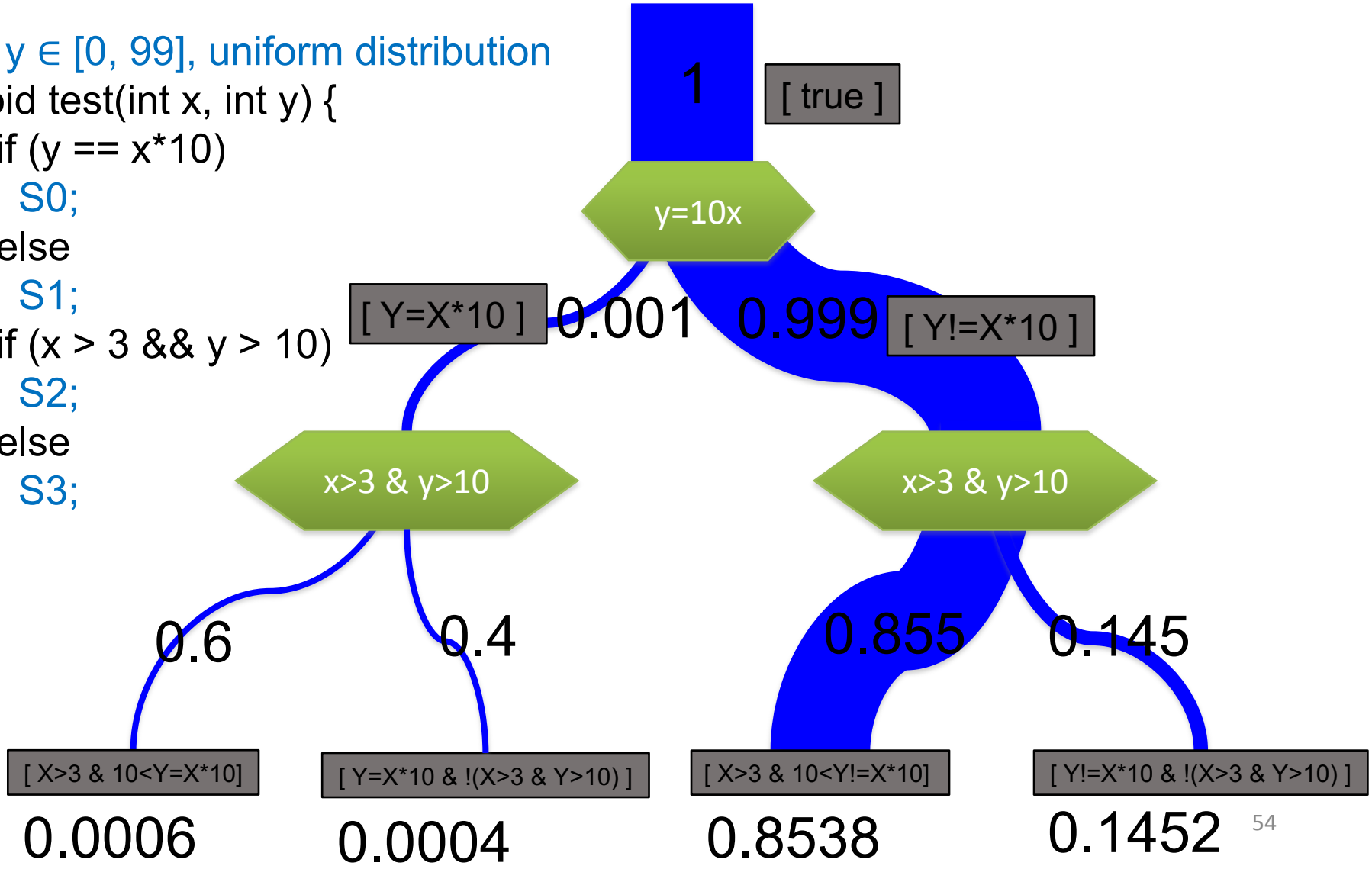
```
x, y ∈ [0, 99]
void test(int x, int y) {
   if (y == x*10)
     S0;
   else
     S1;
   if (x > 3 && y > 10)
     S2;
   else
     S3;
}
```



Probabilities

x, y ∈ [0, 99], uniform distribution

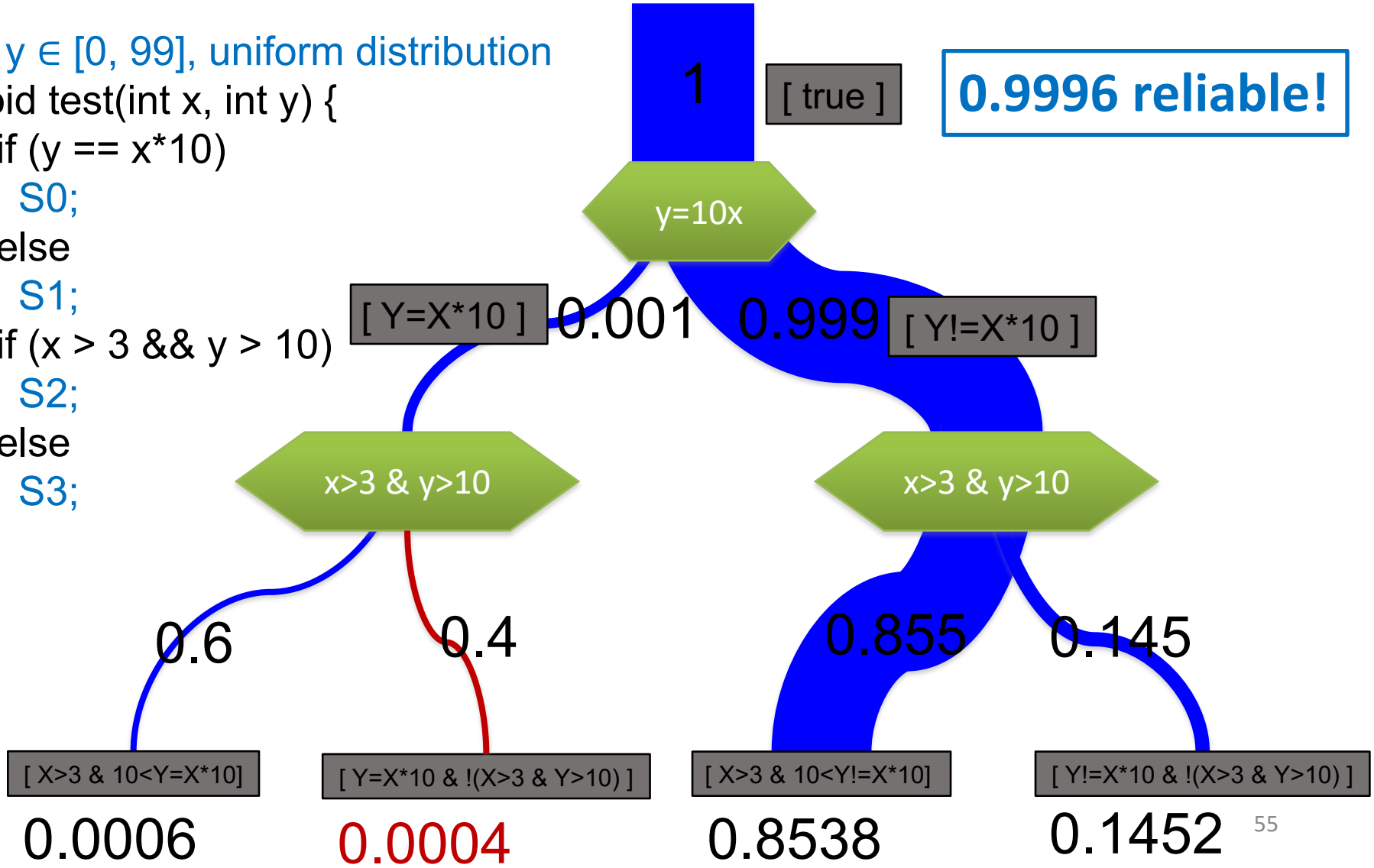
```
void test(int x, int y) {
   if (y == x*10)
     S0;
   else
     S1;
   if (x > 3 && y > 10)
     S2;
   else
     S3;
}
```



Probabilities

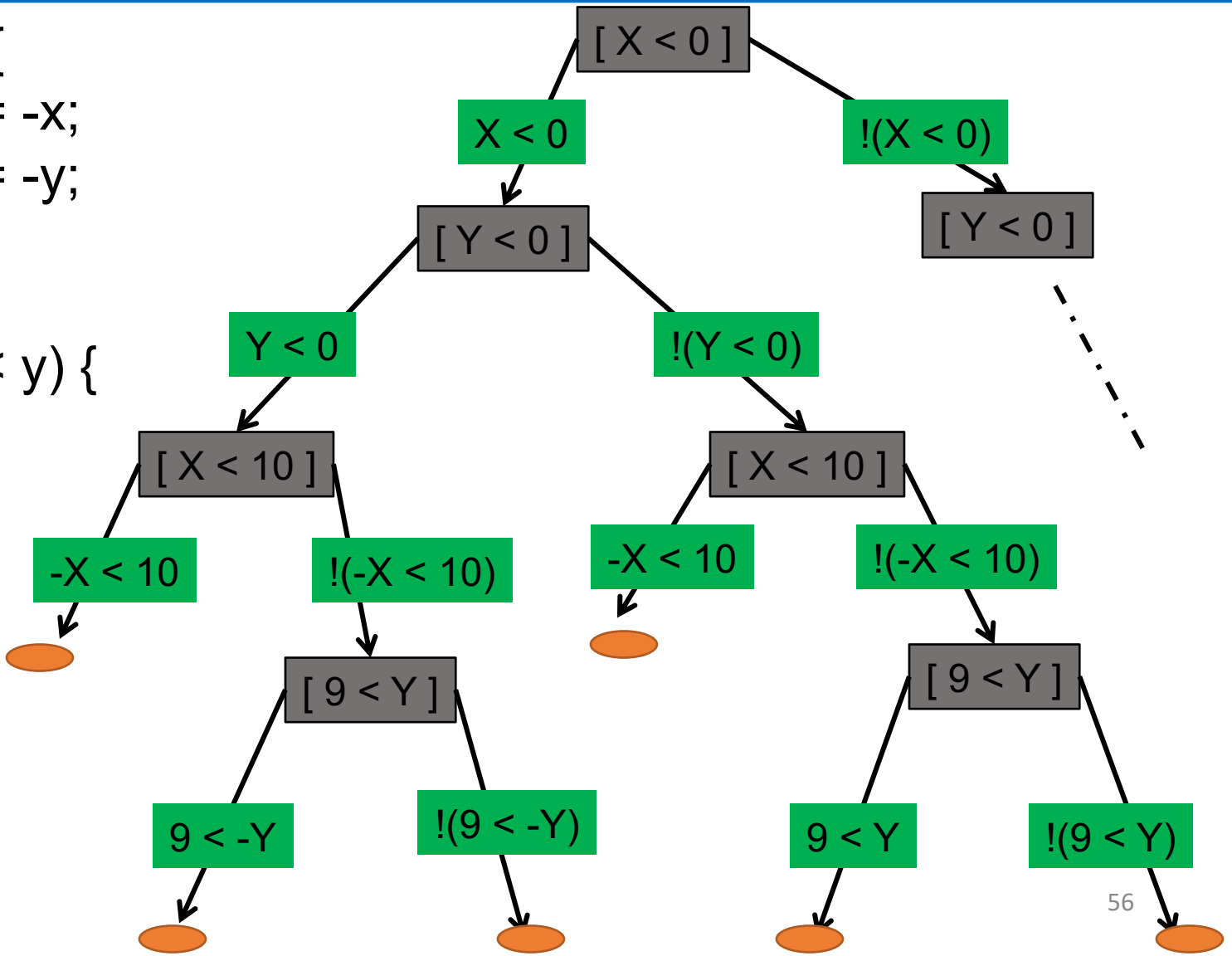
x, y ∈ [0, 99], uniform distribution

```
void test(int x, int y) {
   if (y == x*10)
     S0;
   else
     S1;
   if (x > 3 && y > 10)
     S2;
   else
     S3;
}
```

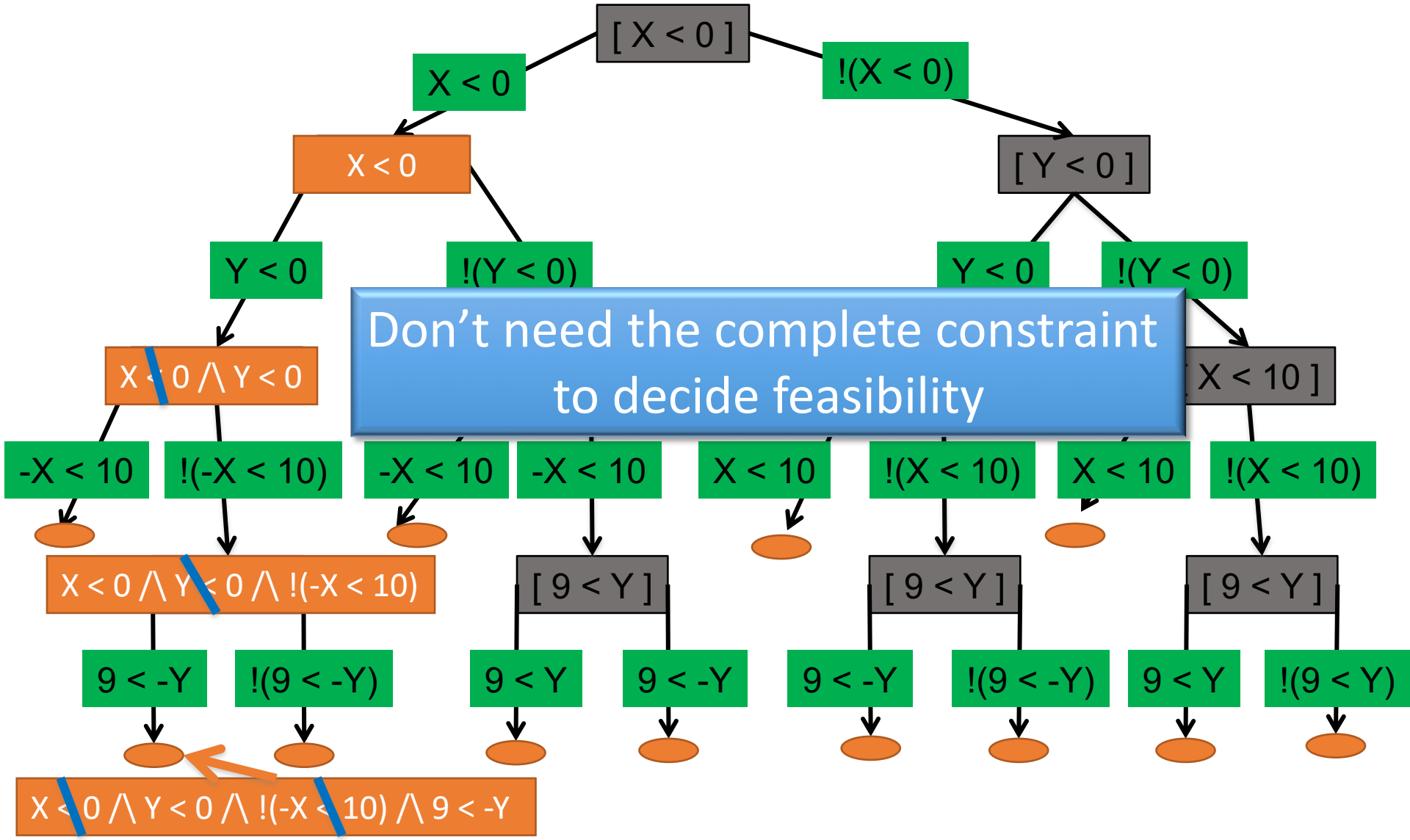


Reduce, Reuse and Recycle Constraints

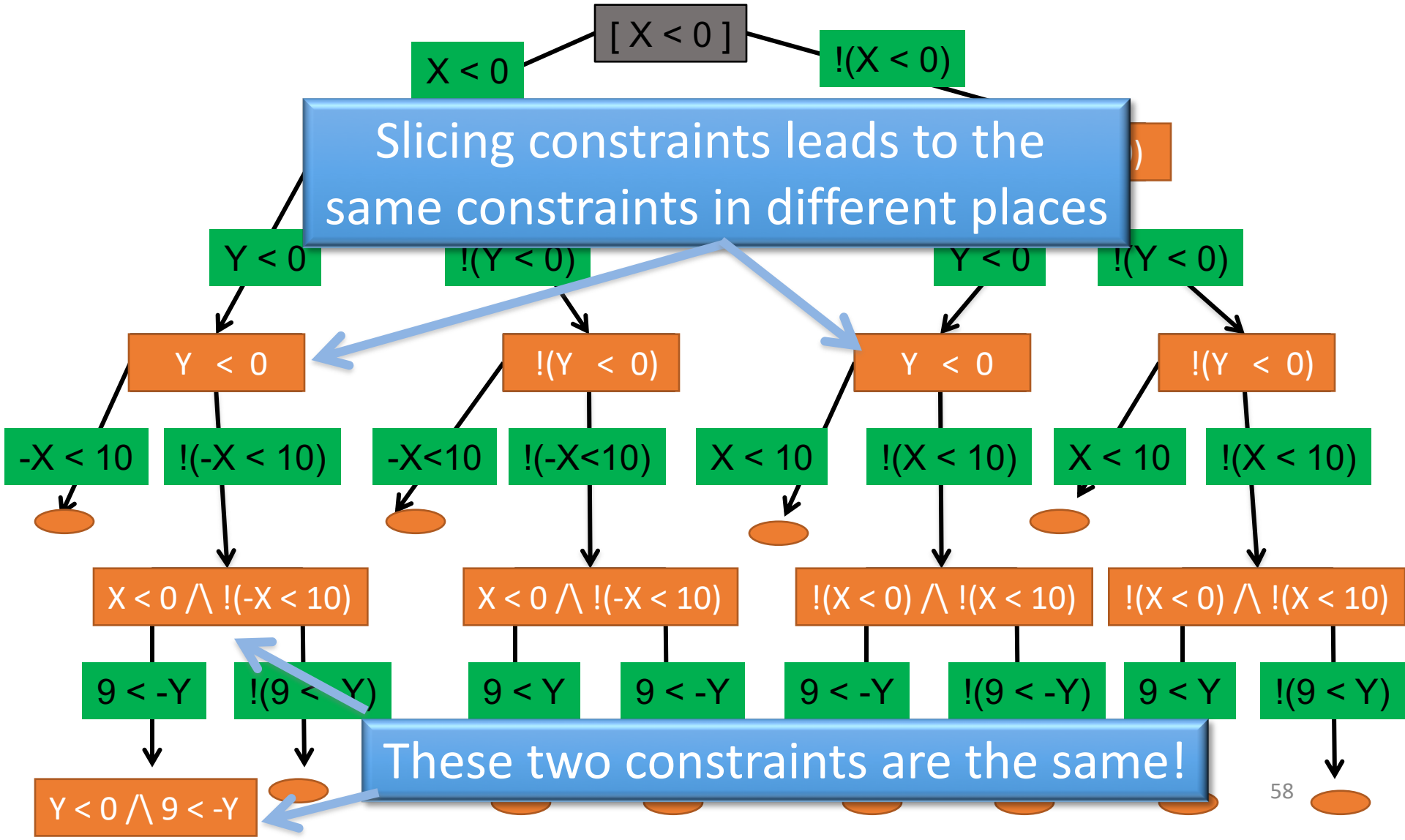
```
int m(int x,y) {
   if (x < 0) x = -x;
   if (y < 0) y = -y;
   if (x < 10) {
      return 1;
   } else if (9 < y) {
      return -1;
   } else {
      return 0;
   }
}
```



Reduce, Reuse and Recycle Constraints



Reduce, Reuse and Recycle Constraints



Canonization of Constraints

$X < 0 \wedge !(-X < 10)$

$X < 0 \wedge -X >= 10$

$X < 0 \wedge X <= -10$

$X + 1 <= 0 \wedge X + 10 <= 0$

$Y < 0 \wedge 9 < -Y$

$Y < 0 \wedge Y < -9$

$Y < 0 \wedge Y + 9 < 0$

$Y + 1 <= 0 \wedge Y + 10 <= 0$

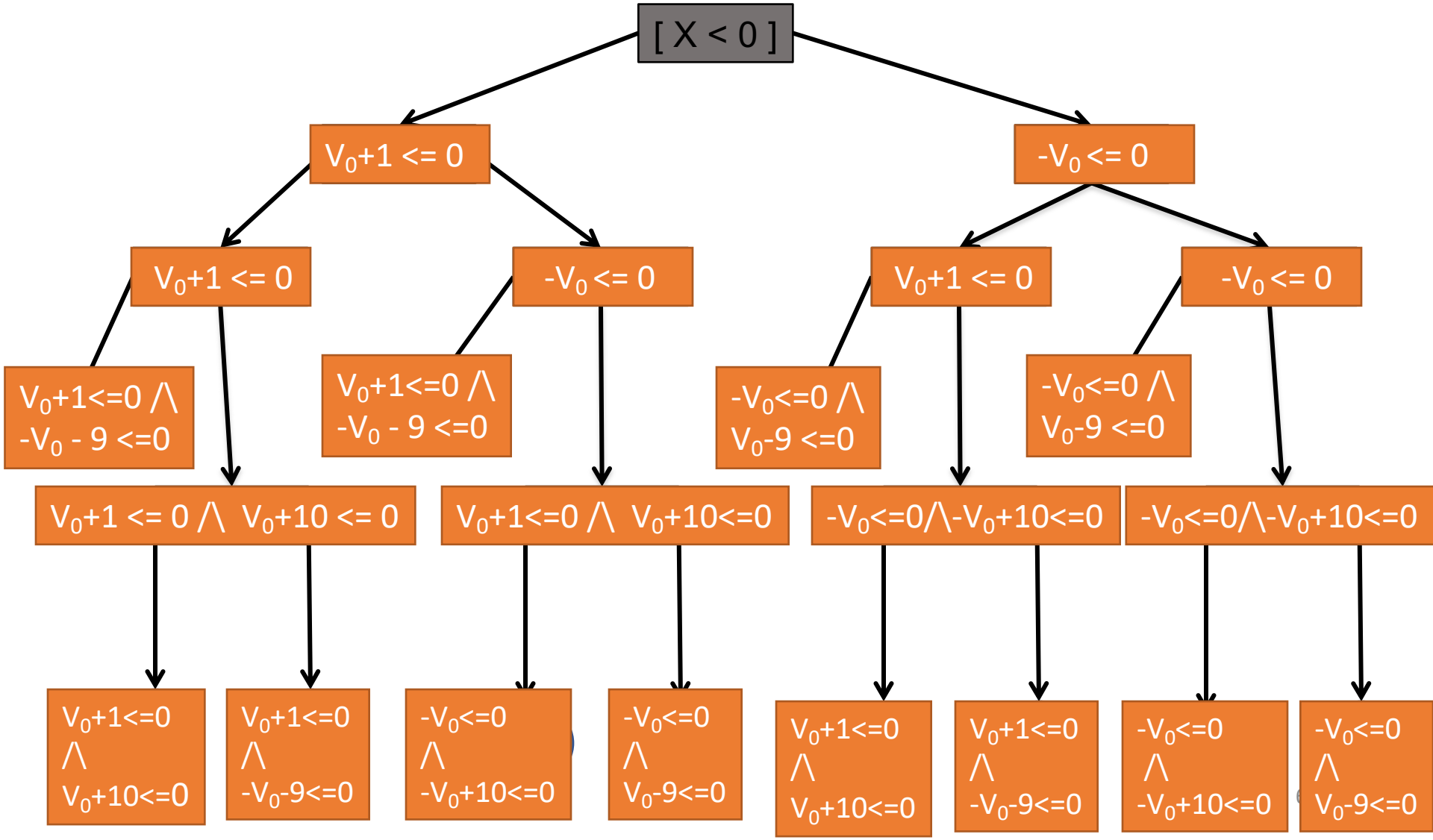
$V_0 + 1 <= 0 \wedge V_0 + 10 <= 0$

Canonical Form

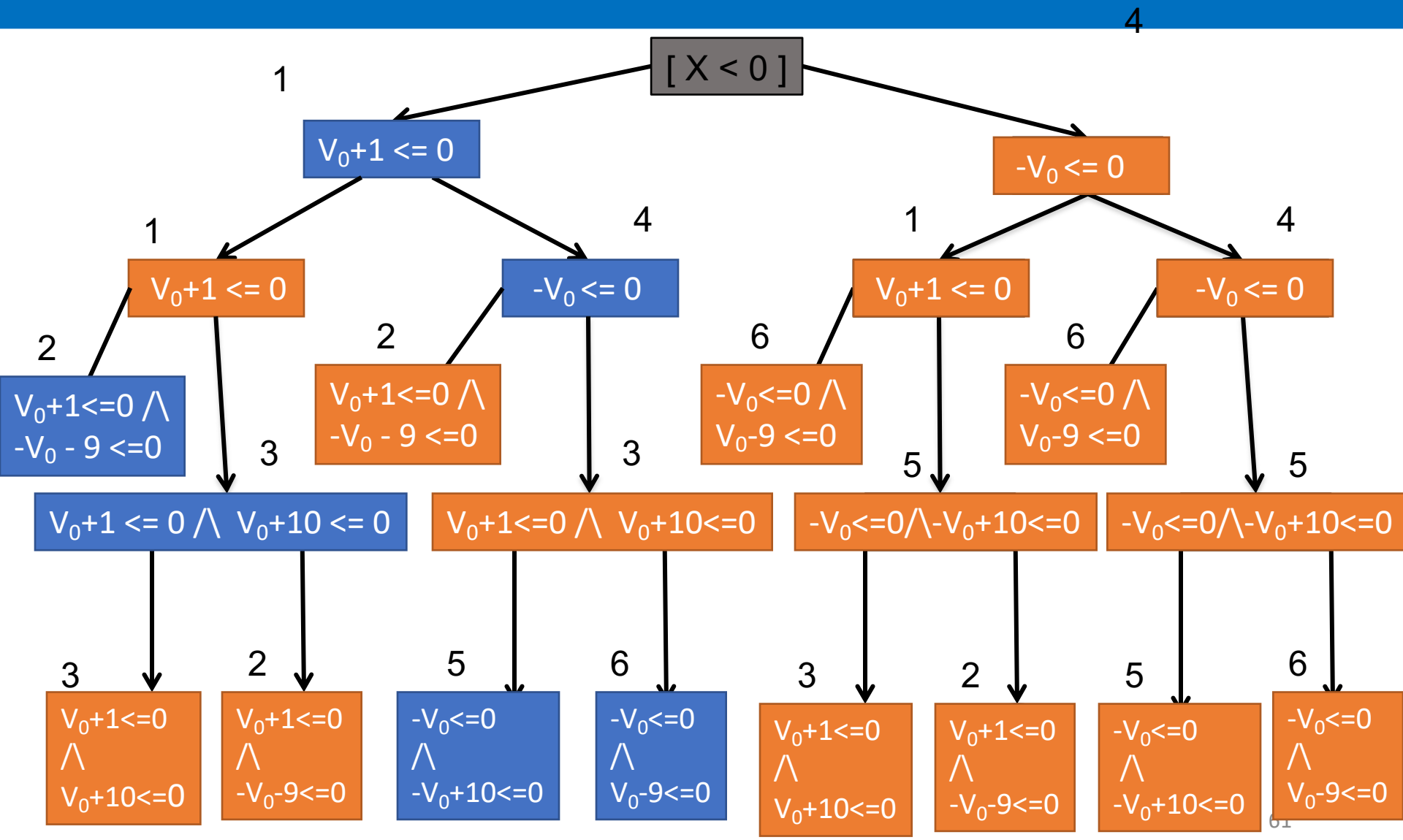
$$ax + by + cz + \ldots + k \{<=, =, !=\} 0$$

- Scale by -1 to transform > and >= to < and <=
- Add 1 to transform < to <=

Canonization of Constraints



Reduce, Reuse and Recycle Constraints



Reduce, Reuse and Recycle Constraints

```
int m(int x,y) {
   if (x < 0) x = -x;
   if (y < 0) y = -y;
   if (x < 10) {
      return 1;
   } else if (10 < y) {
      return -1;
   } else {
      return 0;
   }
}
```

Only the last 8 constraints are changed in the symbolic execution tree and 4 of them are reused!

Reusing the stored results from the first analysis eliminates 14 decision procedure calls!

Reading Materials

- L. A. Clarke. A system to generate test data and symbolically execute programs. IEEE Trans. Softw. Eng., 2(3):215–222, 1976.
- J. C. King. Symbolic execution and program testing. Commun. ACM, 19(7):385–394, 1976.
- C. Cadar, D. Dunbar, and D. Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In OSDI, pages 209–224, 2008.
- C. S. Pa˘sa˘reanu, W. Visser, D. Bushnell, J. Geldenhuys, P. Mehlitz, and N. Rungta. Symbolic PathFinder: Integrating symbolic execution with model checking for Java bytecode analysis. Automat. Softw. Eng., 20(3):391–425, 2013.
- J. Geldenhuys, M. B. Dwyer, and W. Visser. Probabilistic symbolic execution. In ISSTA, pages 166–176, 2012.
- J. A. De Loera, B. Dutra, M. Ko¨Ppe, S. Moreinis, G. Pinto, and J. Wu. Software for exact integration of polynomials over polyhedra. Comput. Geom. Theory Appl., 46(3):232–252, 2013.
- M. Borges, A. Filieri, M. D'Amorim, and C. S. Pa˘sa˘reanu. Iterative distribution-aware sampling for probabilistic symbolic execution. In FSE, 2015.
- L. Luu, S. Shinde, P. Saxena, and B. Demsky. A model counter for constraints over unbounded strings. In PLDI, pages 565–576, 2014.
- A. Filieri, C. S. Pa˘sa˘reanu, and W. Visser. Reliability analysis in symbolic pathfinder. In ICSE, pages 622–631, 2013.
- W. Visser, J. Geldenhuys, and M. B. Dwyer. Green: reducing, reusing and recycling constraints in program analysis. In FSE, 2012.
- P. Godefroid, N. Klarlund, and K. Sen. DART: directed automated random testing. In PLDI, pages 213-223, 2005.

Q&A?

Bihuan Chen, Pre-Tenure Assoc. Prof.

bhchen@fudan.edu.cn

https://chenbihuan.github.io