CSCI426/CSCI926
Software Testing and Analysis

Symbolic Execution (Part 2)

Acknowledgement: Slides are adapted from Koushik Sen, Cristian Cadar, Omar Chowdhury, Jeff Foster, Suman Jana and Pezze & Young

Symbolic Execution Limitations

- **Loops and recursions** --- infinite execution tree
- **Path explosion** --- exponentially many paths
- **Heap modeling** --- symbolic data structures and pointers
- **SMT solver limitations** --- dealing with complex path constraints
- **Environment modeling** --- dealing with native/system/library calls/file operations/network events
- **Coverage Problem** --- may not reach deep into the execution tree, specially when encountering loops.

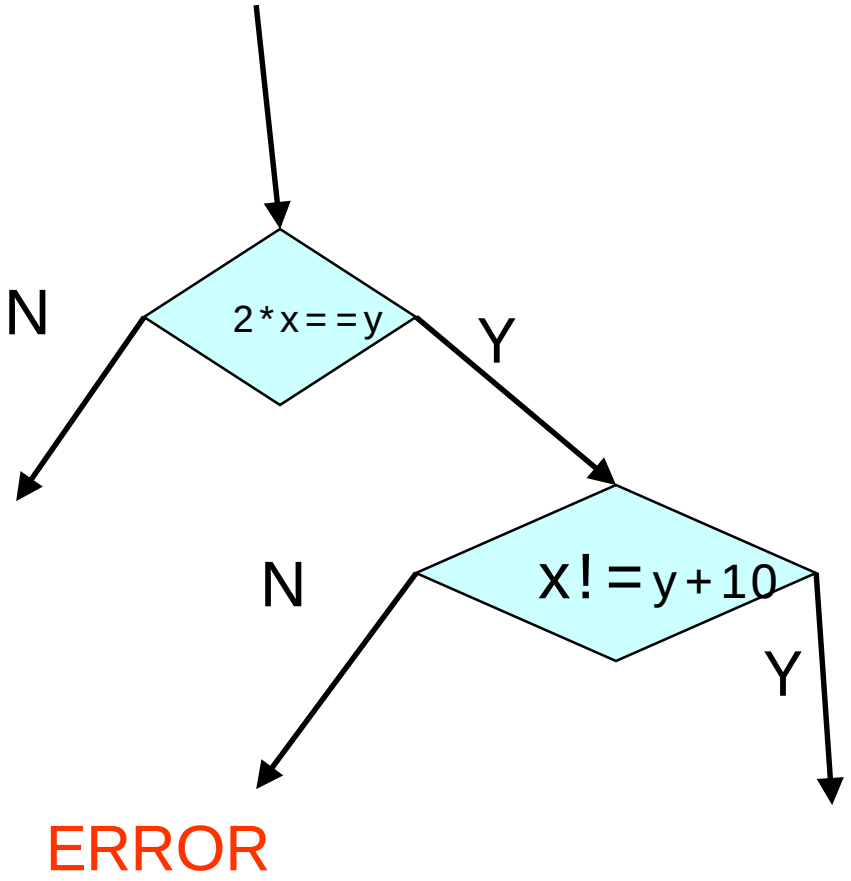
Symbolic Execution Limitations

```
1 void testme inf () {
2       int sum = 0;
3       int N = sym_input();
4       while (N > 0) {
5             sum = sum + N;
6             N = sym_input();
7       }
8       return;
9 }
```

Apply symbolic execution to the above code. How many execution paths are there?

Example of Execution Tree

```
void test_me(int x, int y) {
  if(2*x==y){
    if(x != y+10){
        printf("I am fine here");
    } else {
        printf("I should not reach
    here");
        ERROR;
    }
  }
}
```



Existing Approach I

- Random testing
  - generate random inputs
  - execute the program on generated inputs
- Probability of reaching an error can be astronomically less
  - What is the probability of hitting the ERROR path?

```
test_me(int x){
   if(x==94389){
        ERROR;
   }
}
```

Probability of hitting ERROR = $1/2^{32}$

Existing Approach II

- Symbolic Execution
  - use symbolic values for input variables
  - execute the program symbolically on symbolic input values
  - collect symbolic path constraints
  - use theorem prover to check if a branch can be taken
- Does not scale for large programs

```
test_me(int x){
  if(bbox(x)!=17){
      ERROR;
  } else {
      // OK
  }
}
```

Symbolic execution may not be able to determine whether the error is reachable.

Solution: Concolic Execution

Concolic = **Con**crete + Symb**olic**

Combining Classical Testing with Automatic Program Analysis

Also called **dynamic symbolic execution**

- The intention is to visit deep into the program execution tree
- Program is simultaneously executed with concrete and symbolic inputs
- Start off the execution with a random input
- Specially useful in cases of remote procedure call
- **Concolic execution implementations**: SAGE (Microsoft), CREST

Concolic Execution Steps

1. Generate a random seed input to start execution
2. Concretely execute the program with the random seed input and collect the path constraint - Example: **a && b && c**
3. In the next iteration, negate the last conjunct to obtain the constraint **a && b && !c**
4. Solve it to get input to the path which matches all the branch decisions except the last one

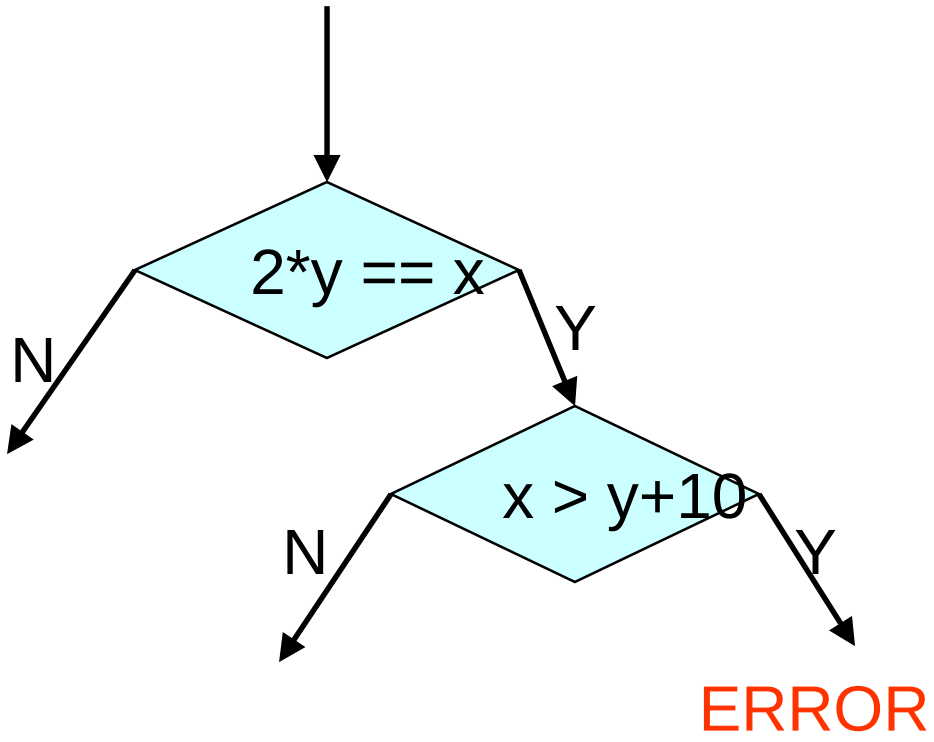
Concolic Testing Approach

```
void testme (int x, int y) {

    z = 2 * y;

    if (z == x) {

        if (x > y+10) {

            ERROR;
        }
    }

}
```

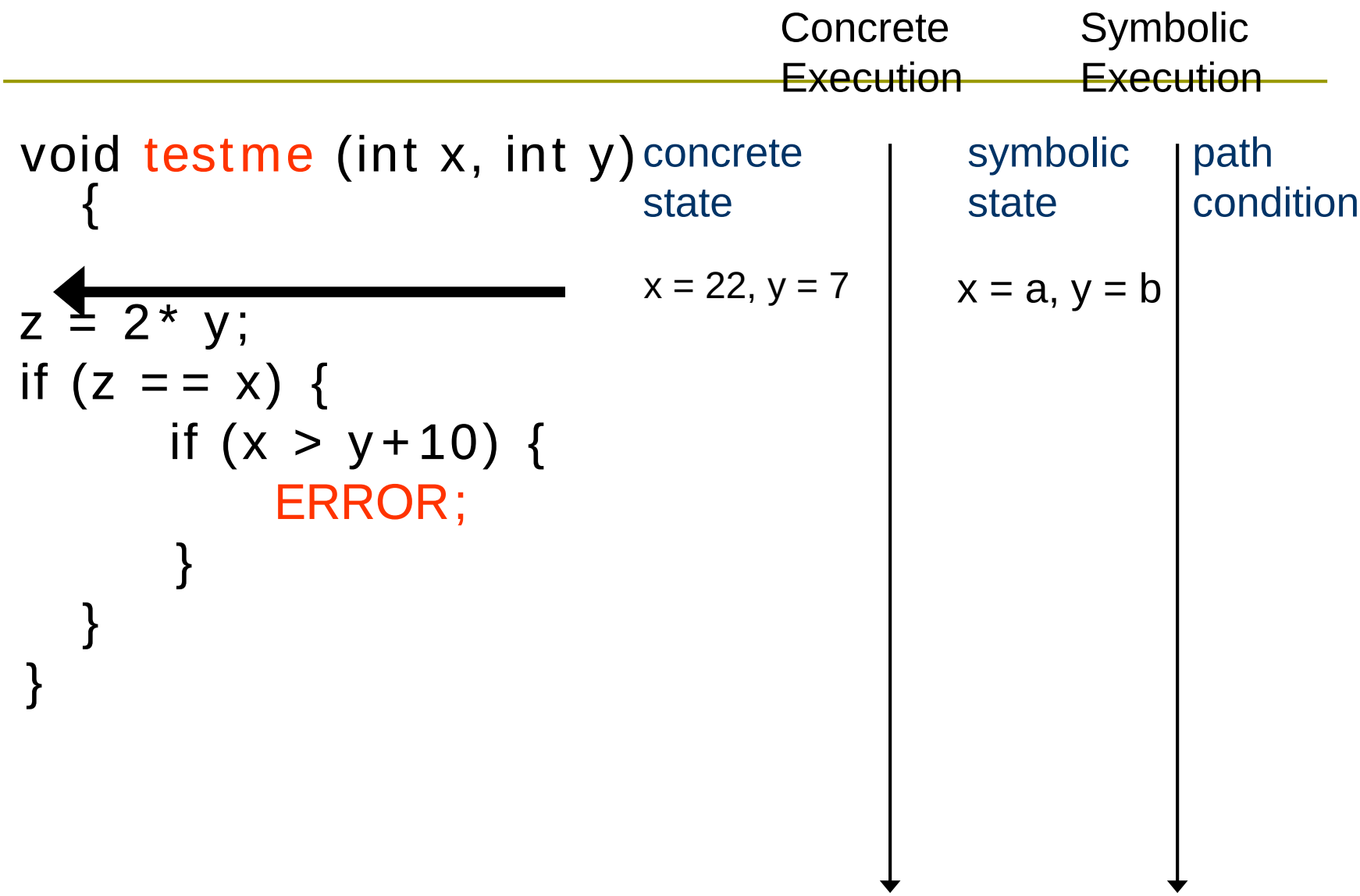
- Random Test Driver:
  - random value for x and y
- Probability of reaching ERROR is extremely low

Example

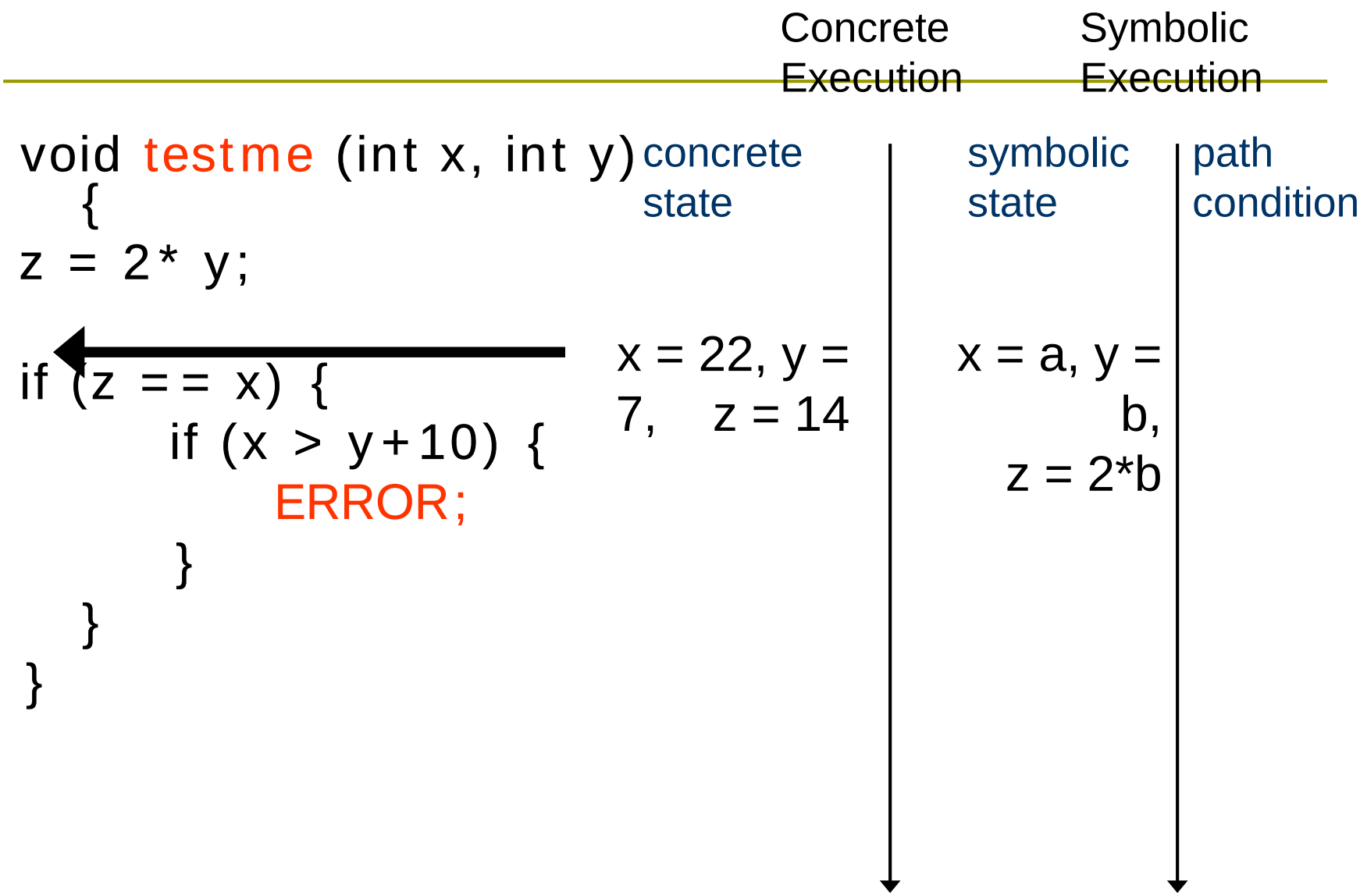
```
void testme (int x, int y)
{
   z = 2*y;
   if (z == x) {
        if (x > y+10) {
             ERROR;
        }
   }
}
```



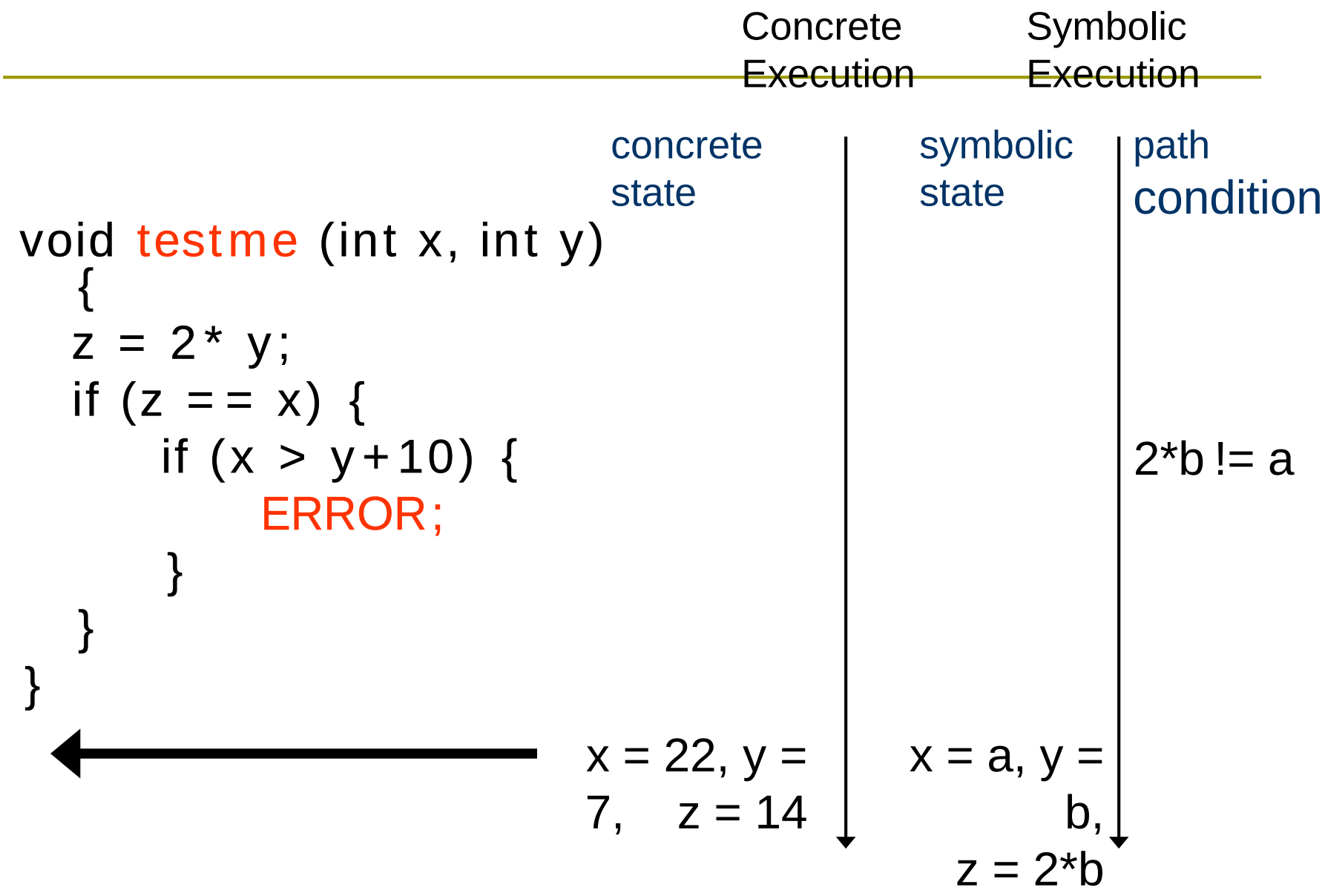
Concolic execution example



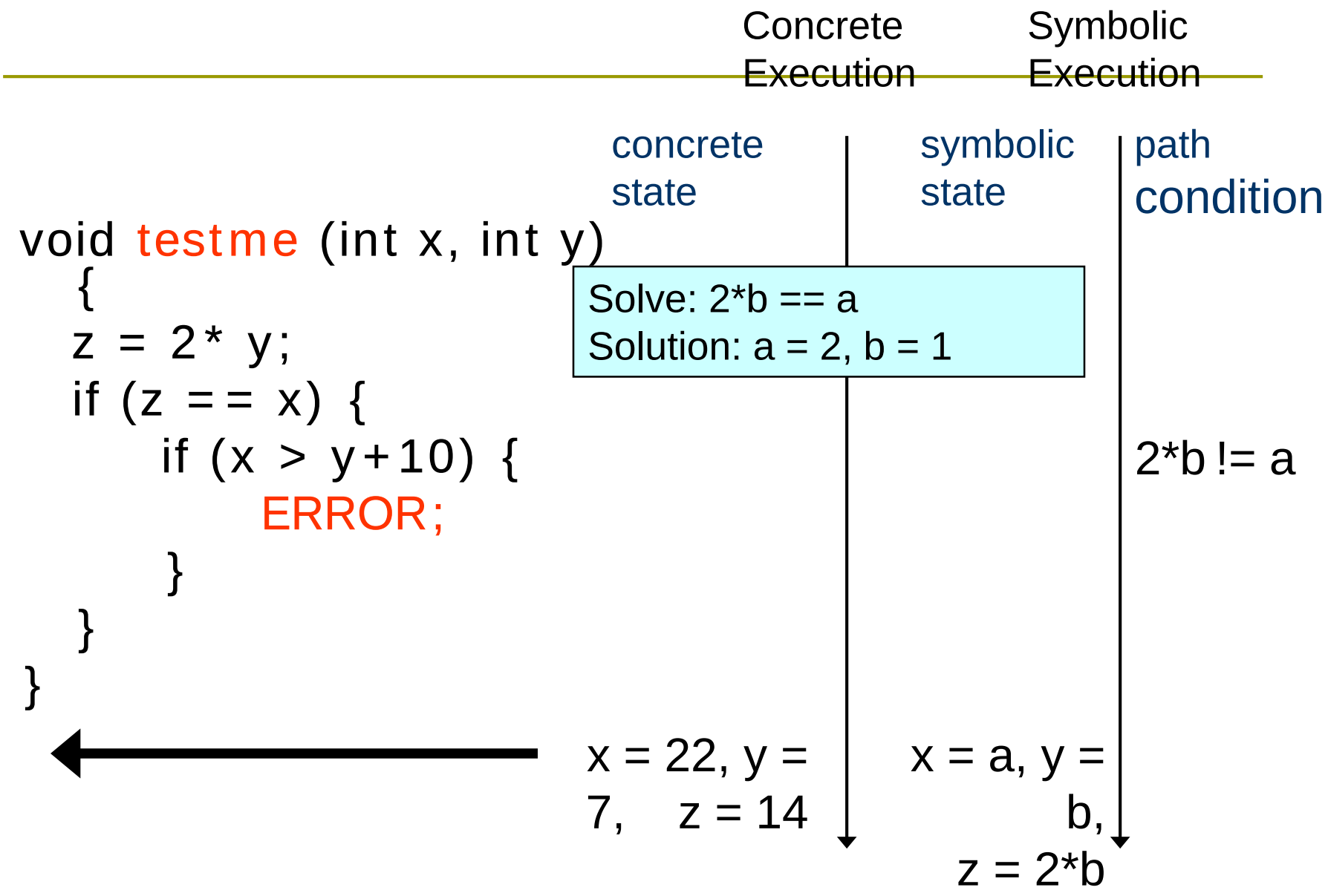
Concolic execution example



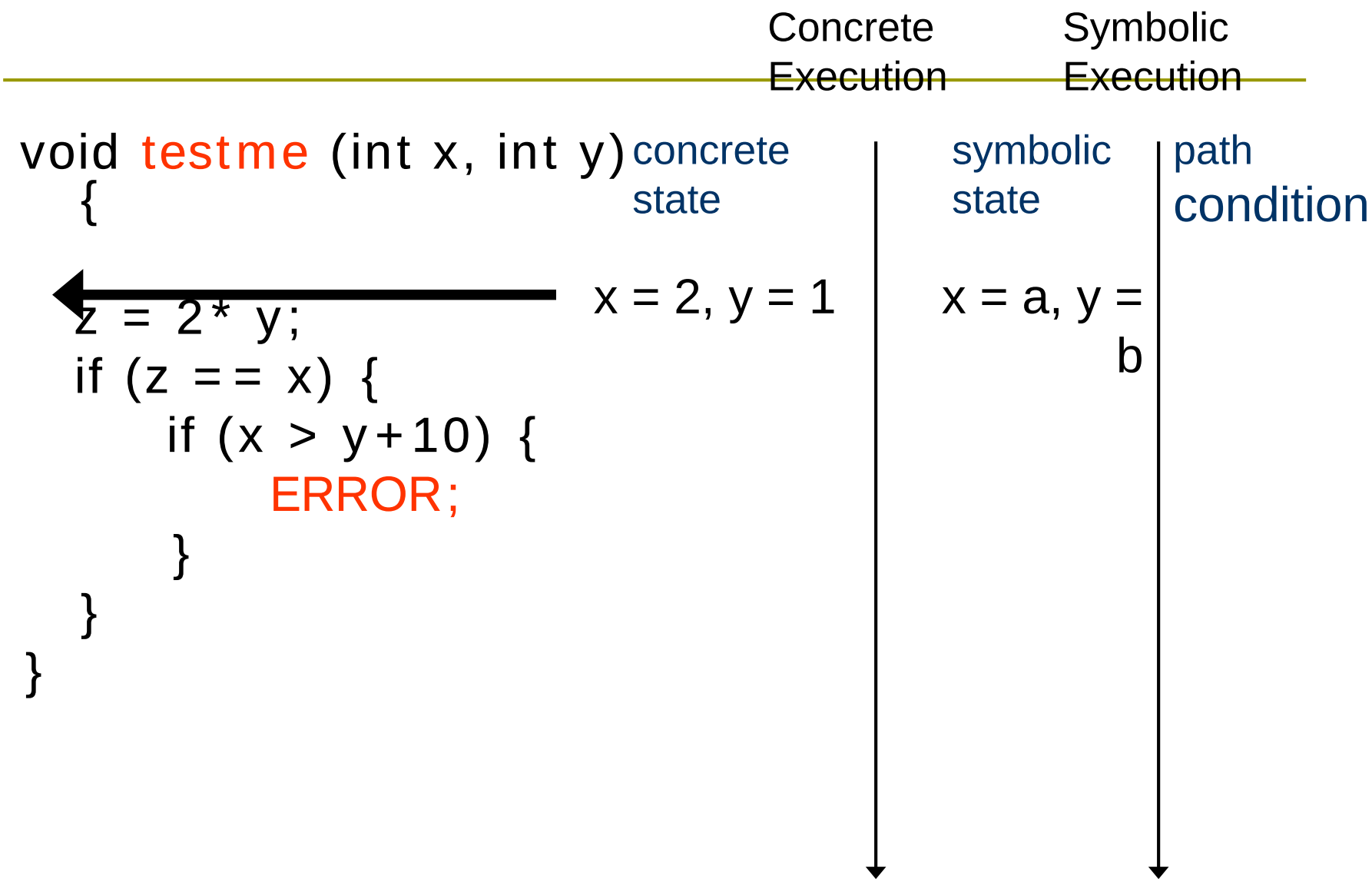
Concolic execution example



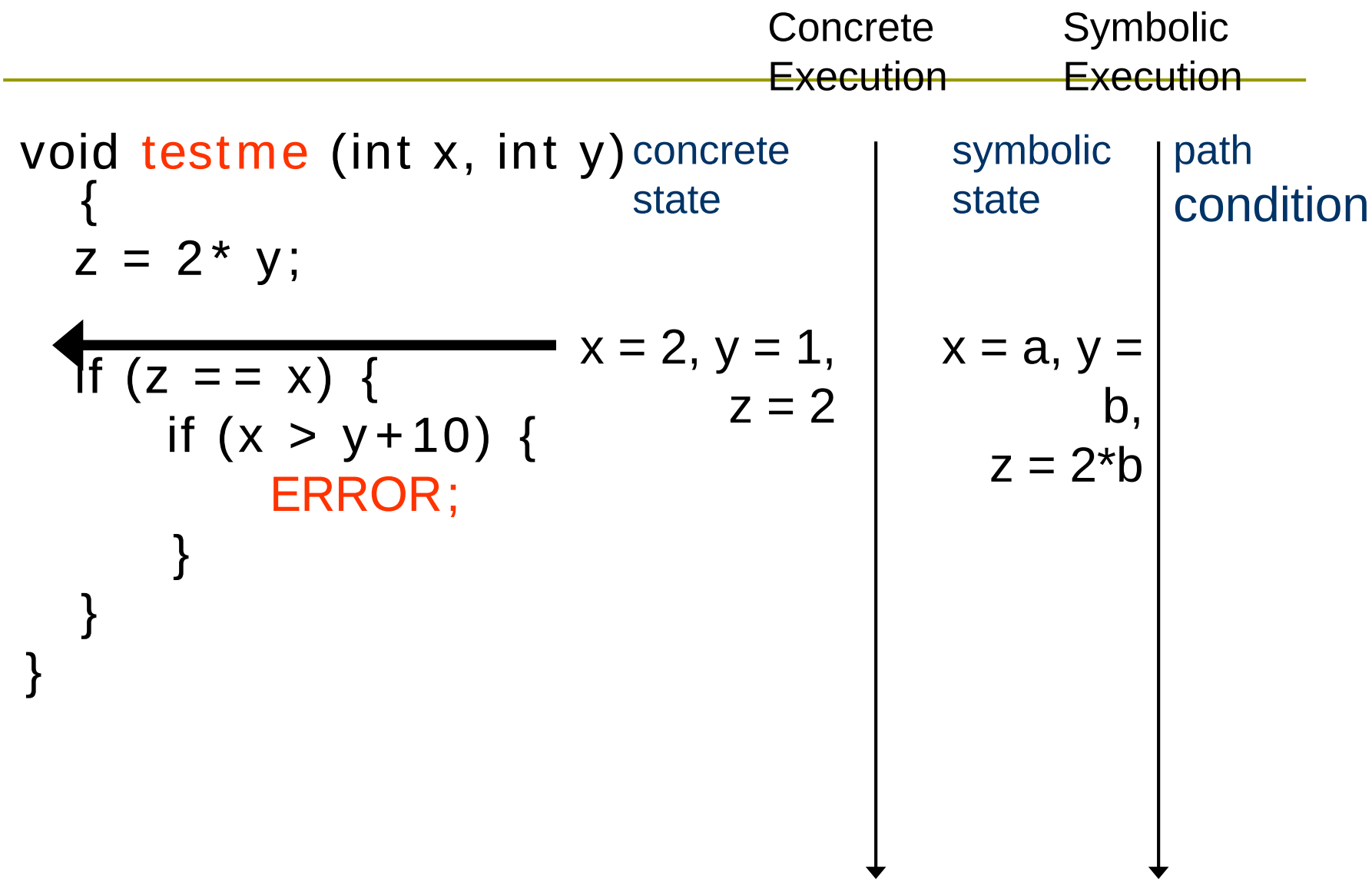
Concolic execution example



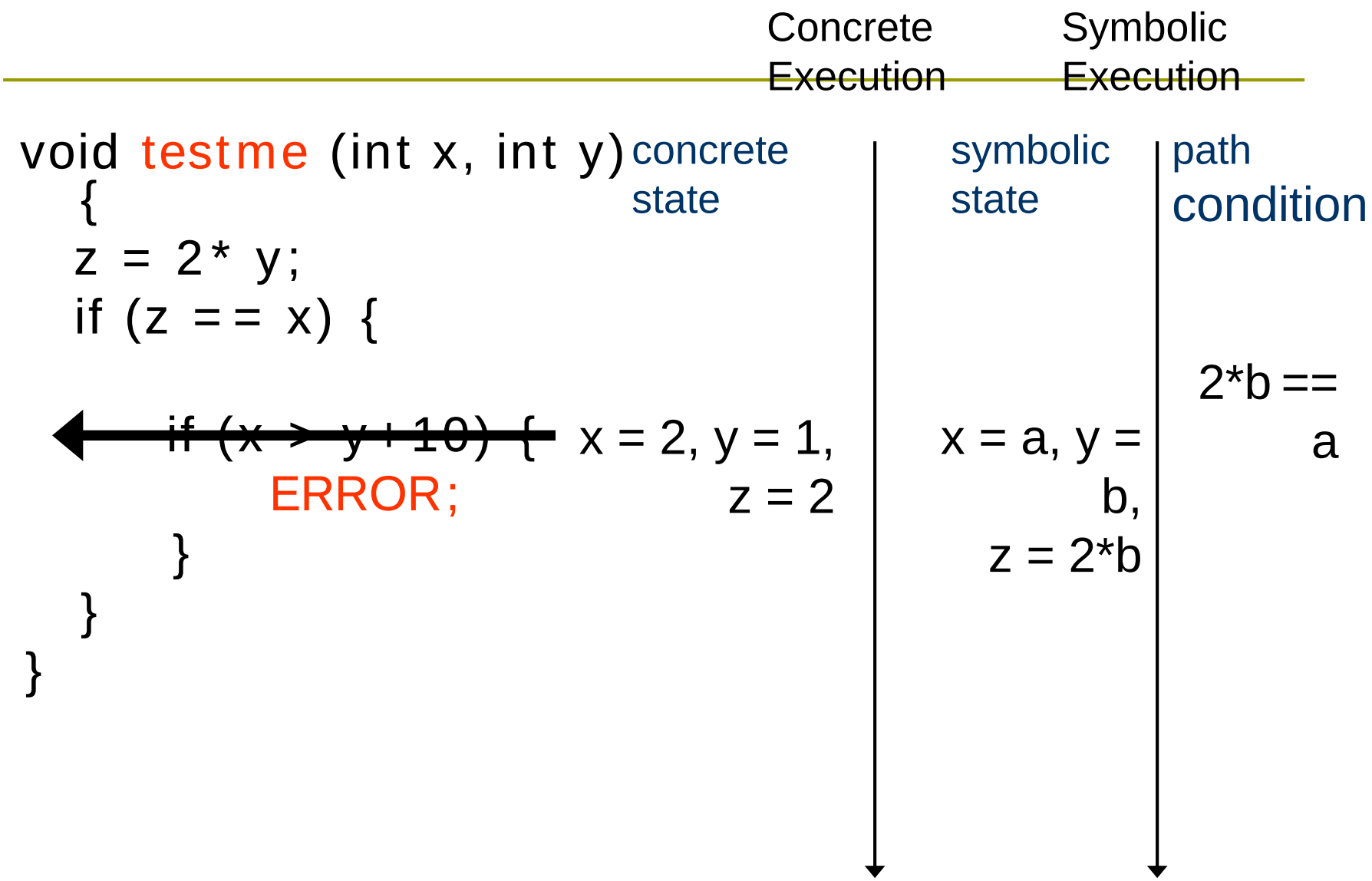
Concolic execution example



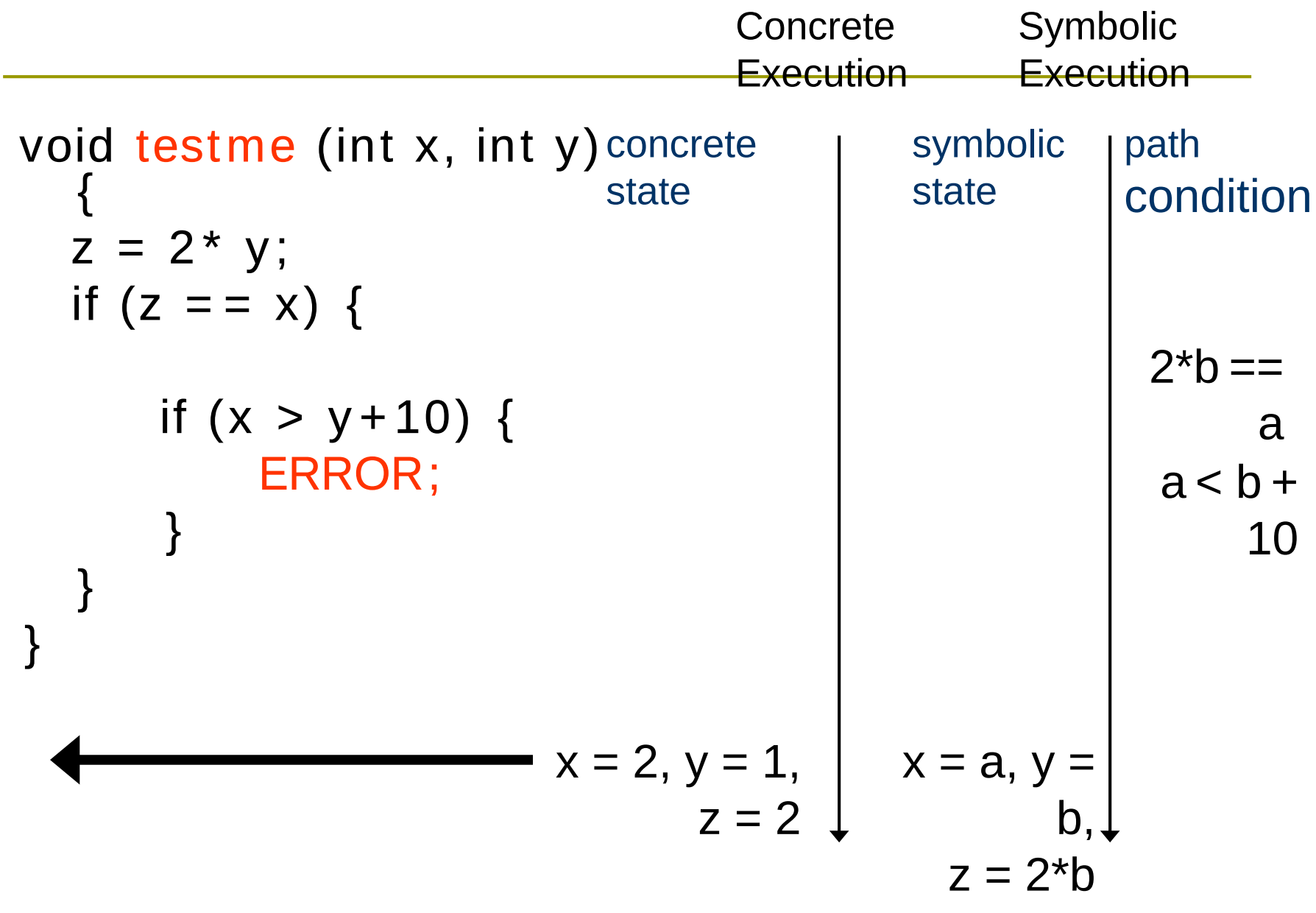
Concolic execution example



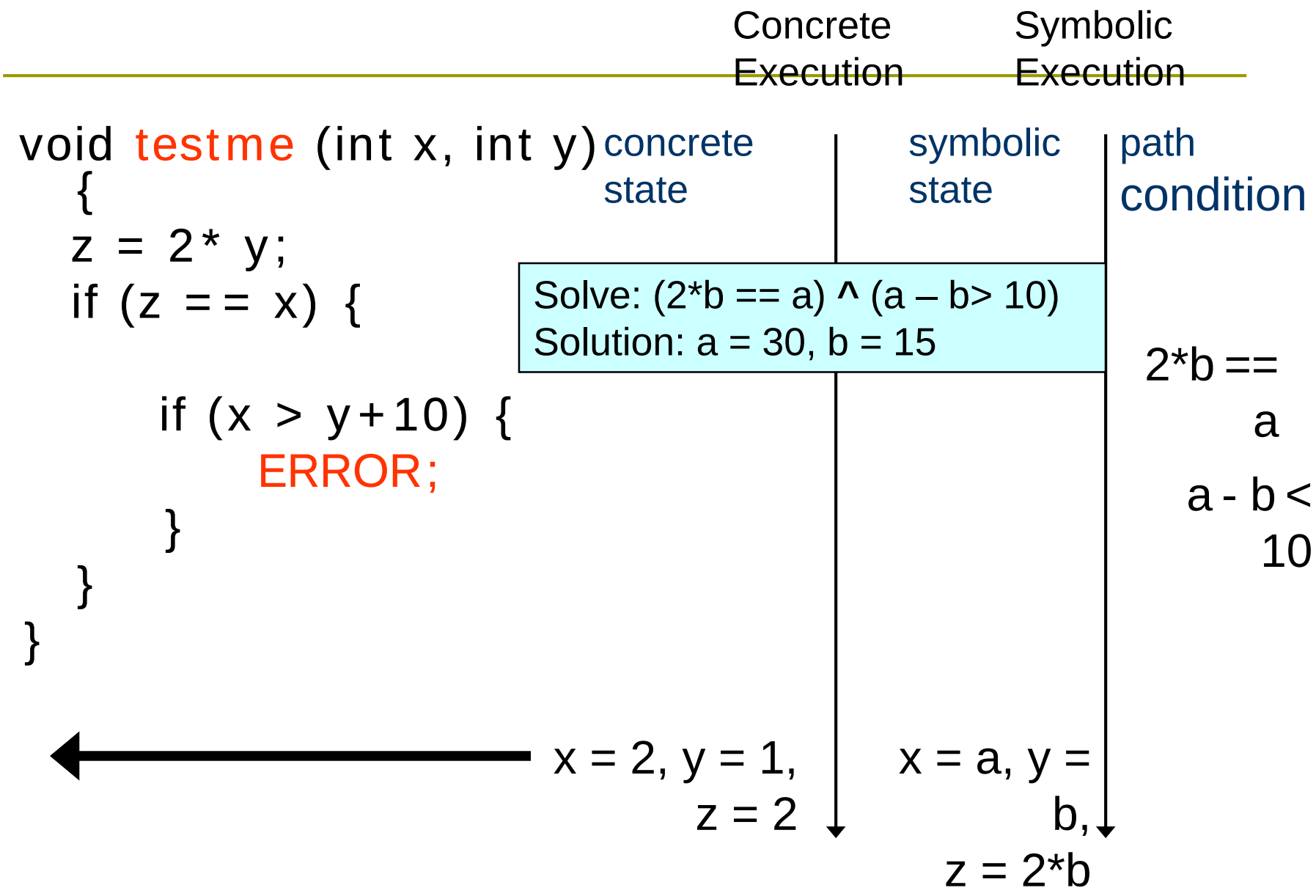
Concolic execution example



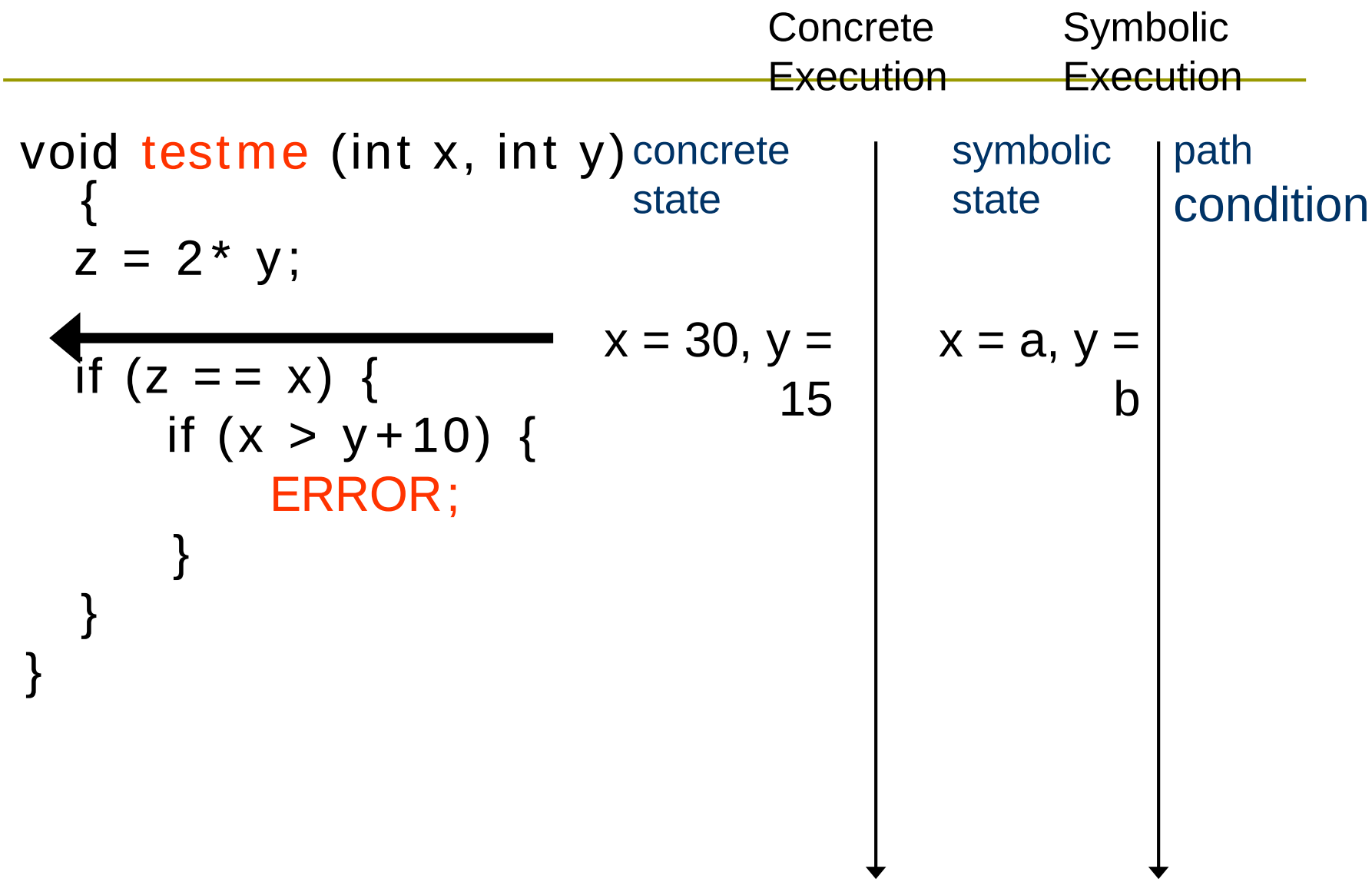
Concolic execution example



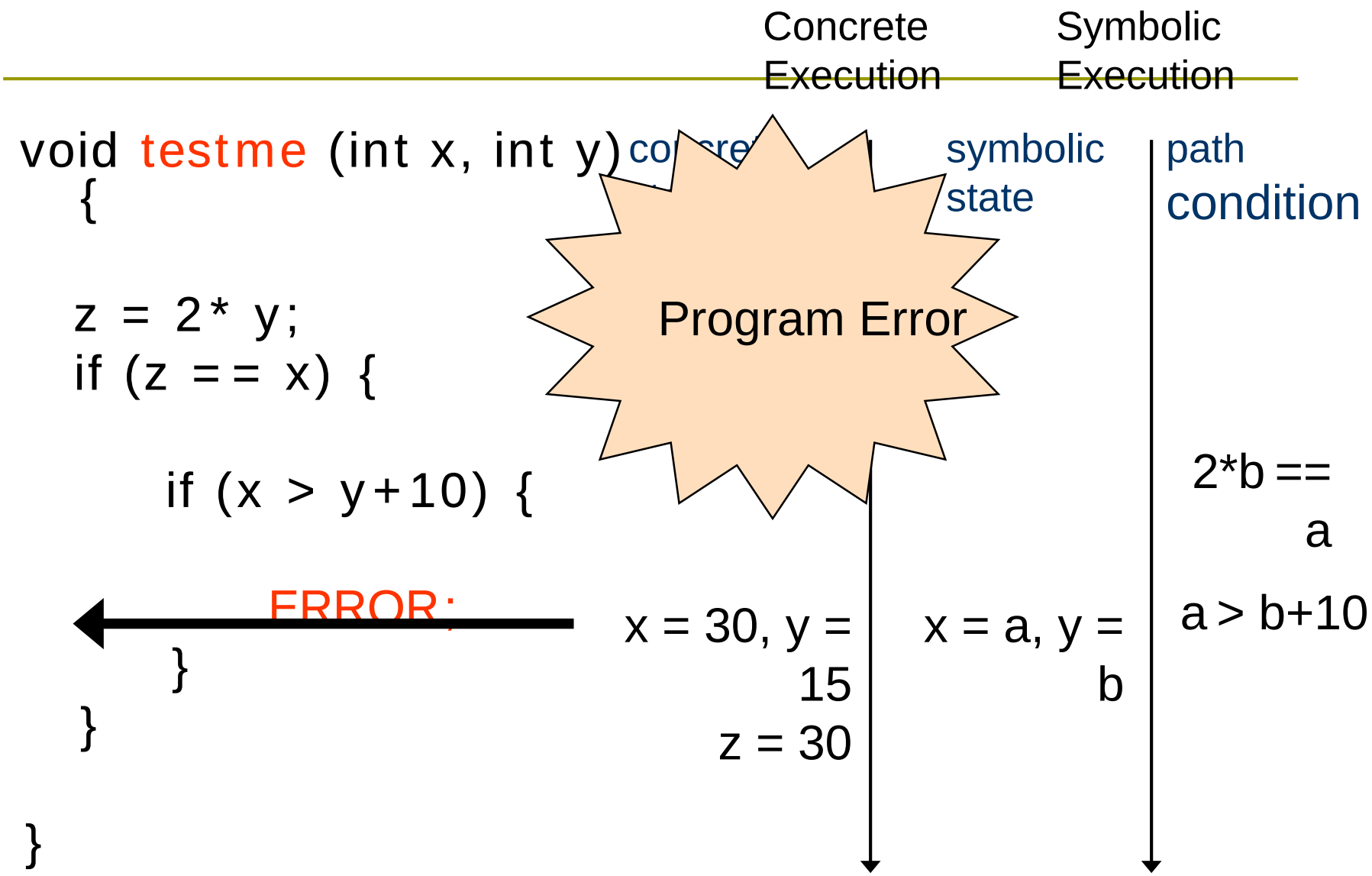
Concolic execution example



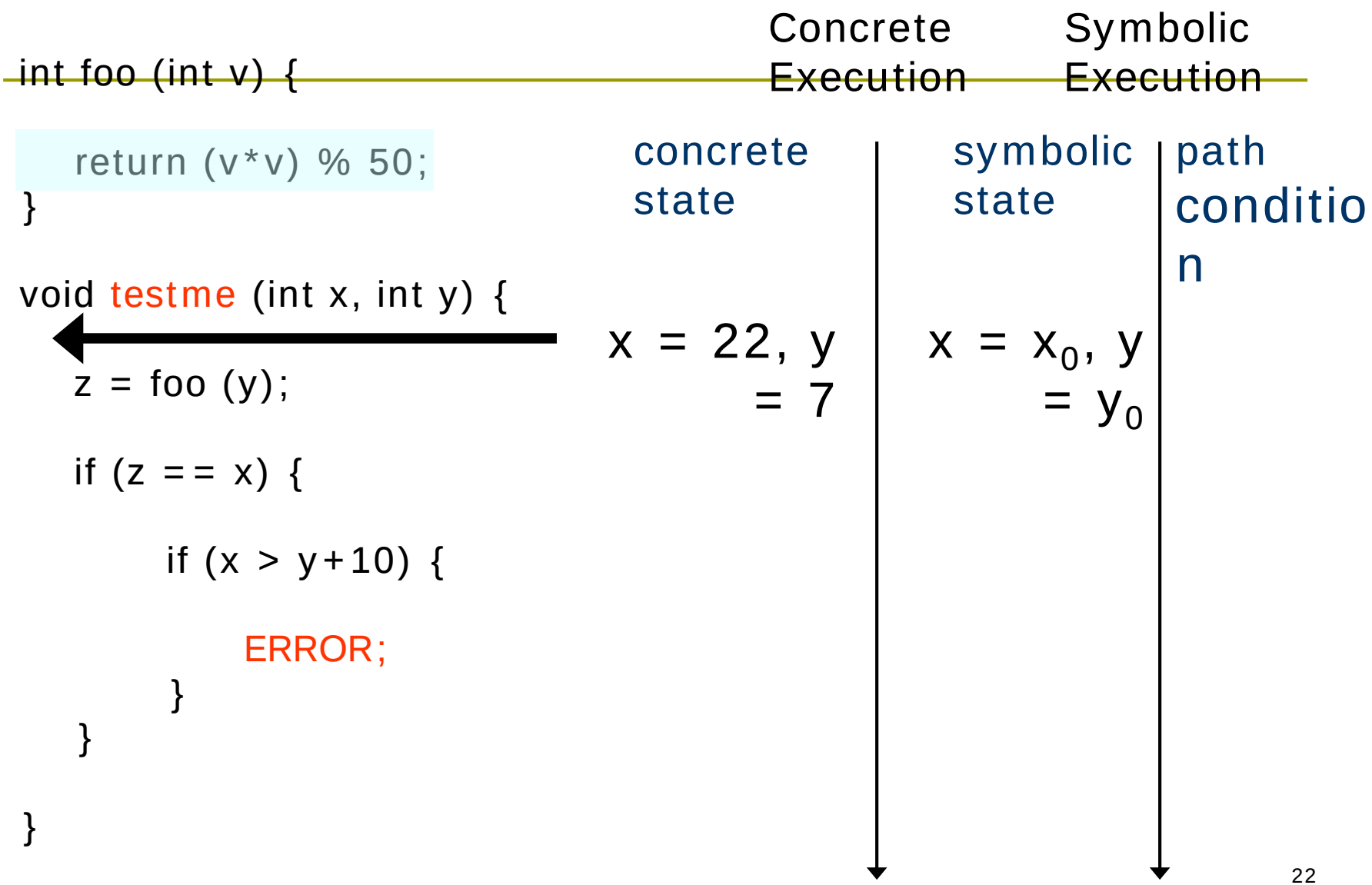
Concolic execution example



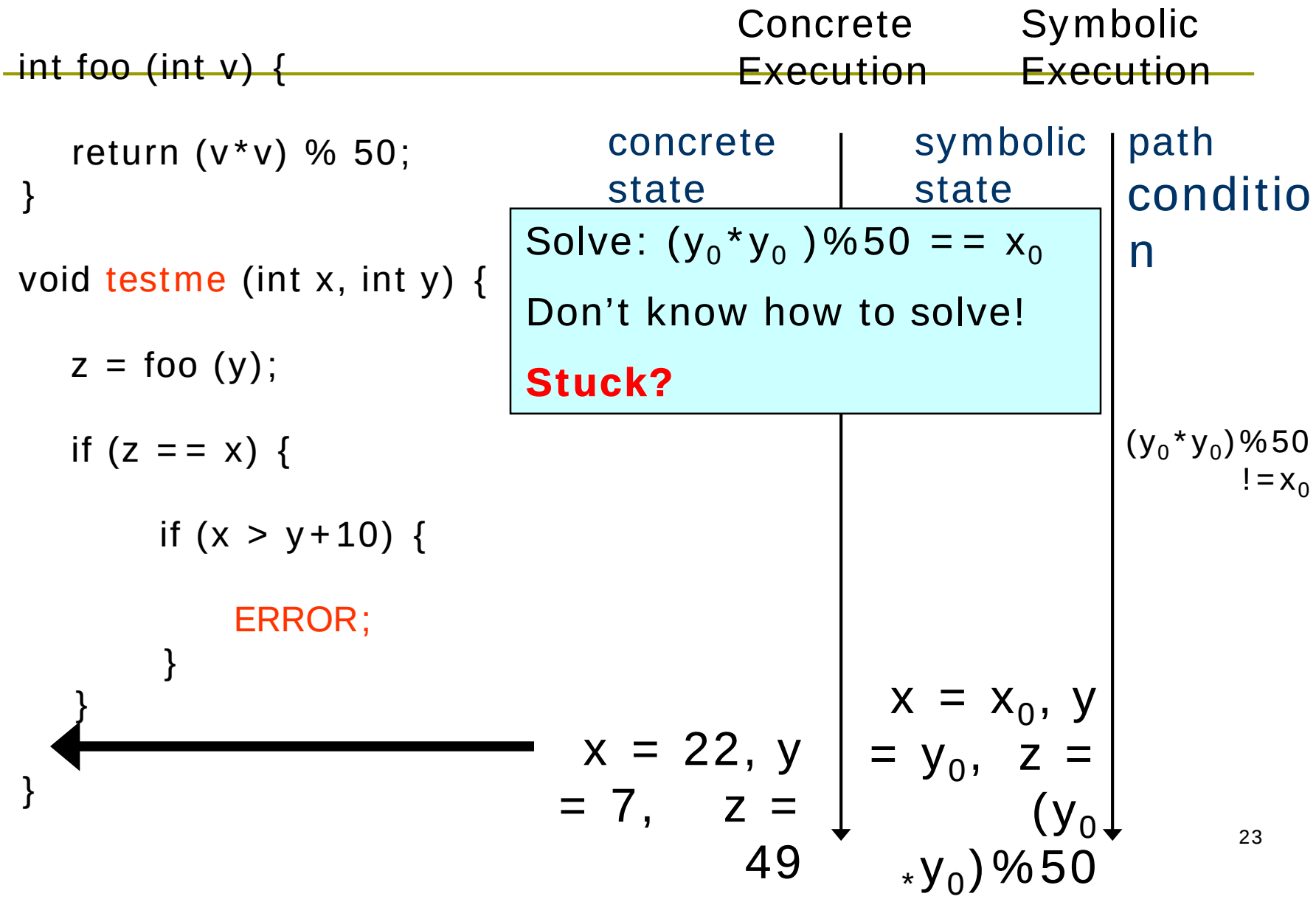
Concolic execution example



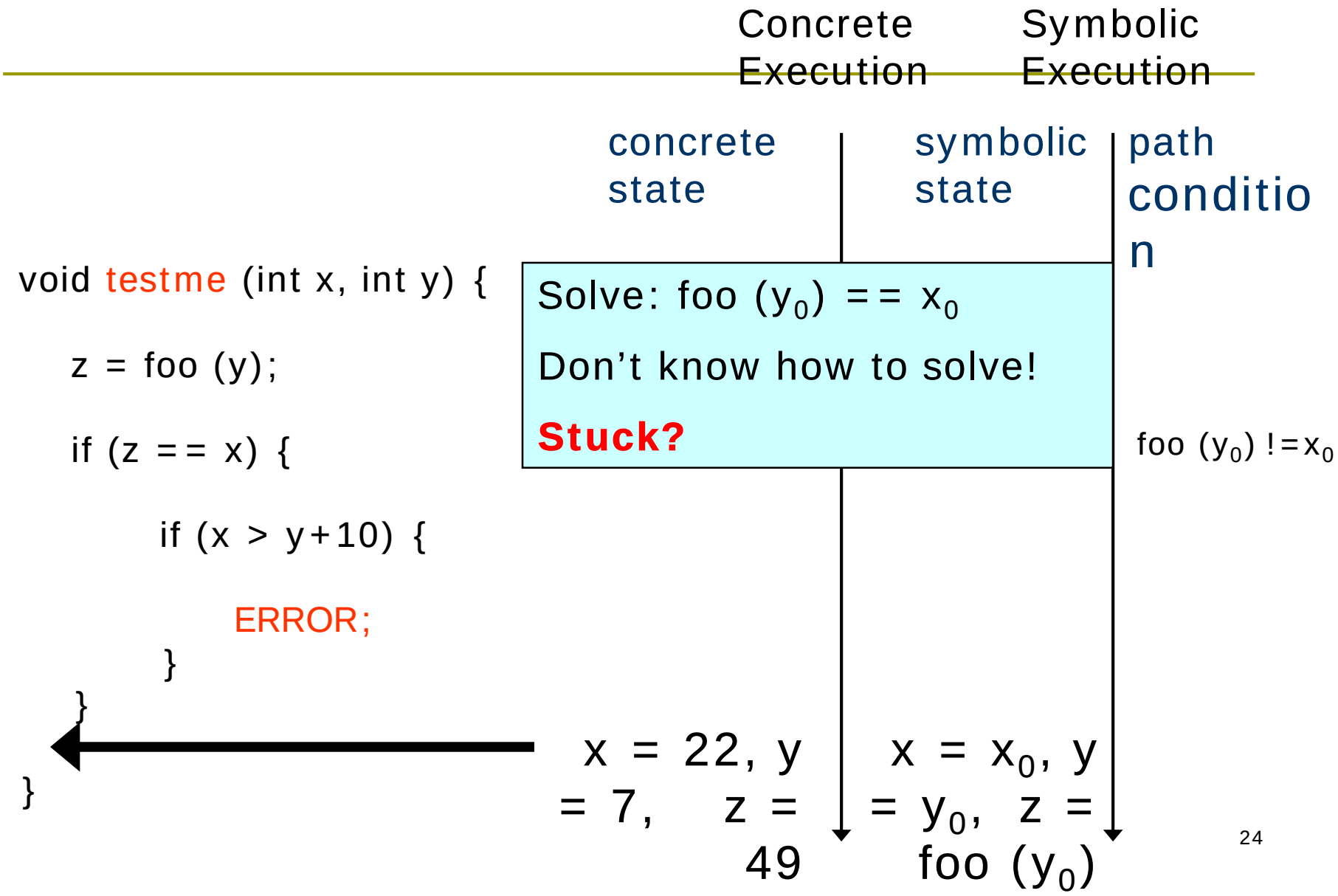
Novelty : Simultaneous Concrete and Symbolic Execution



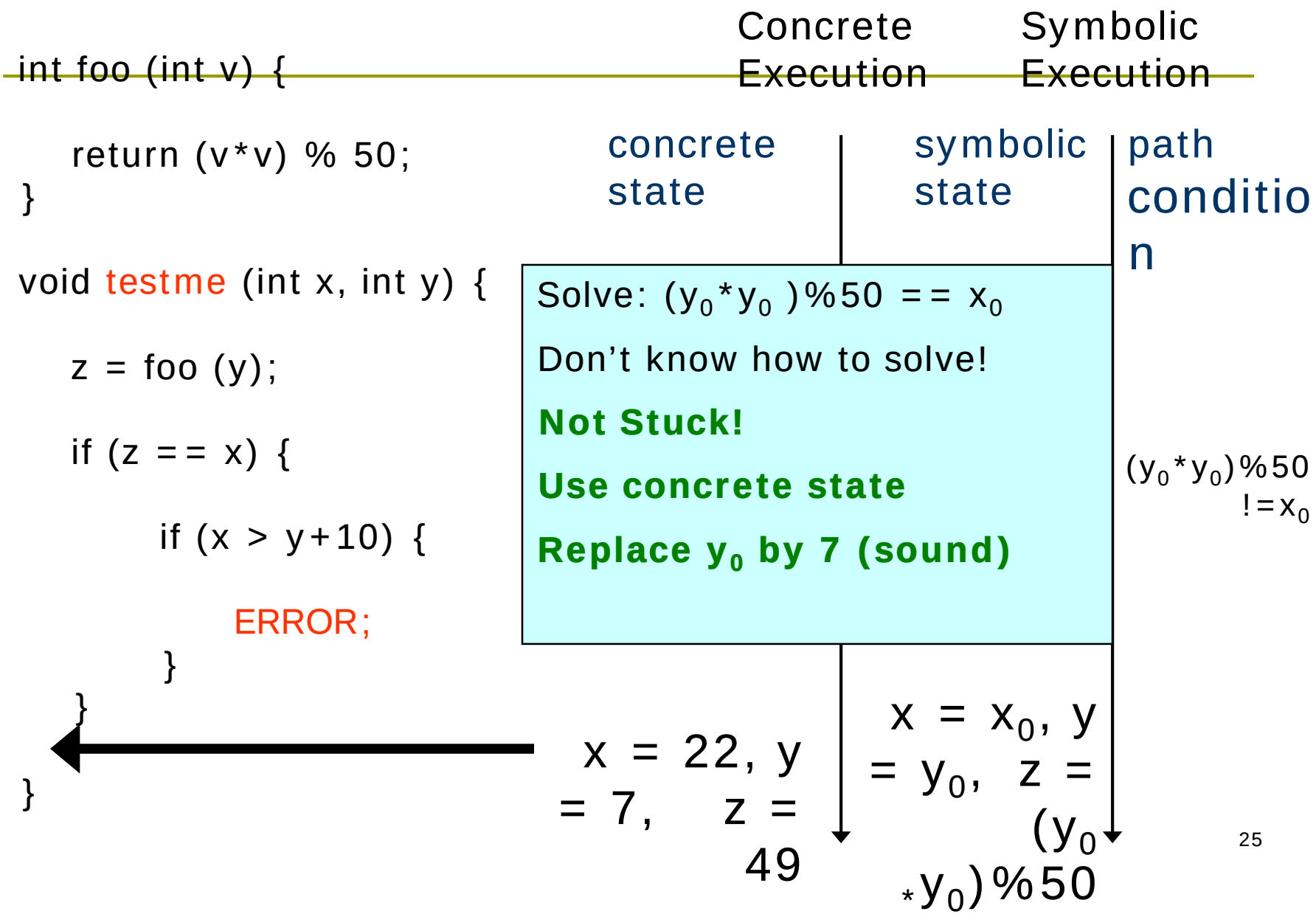
Novelty : Simultaneous Concrete and Symbolic Execution



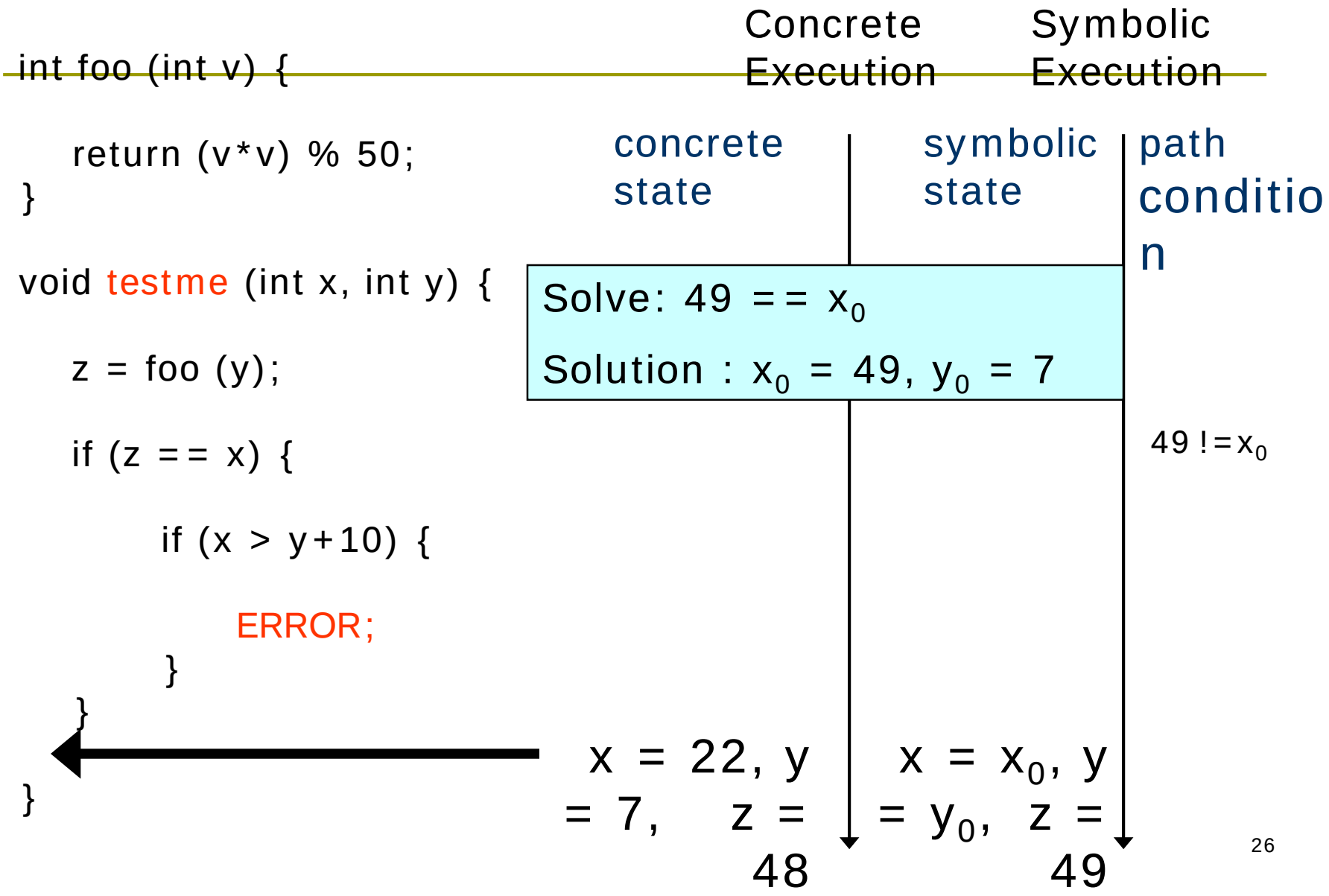
Novelty : Simultaneous Concrete and Symbolic Execution



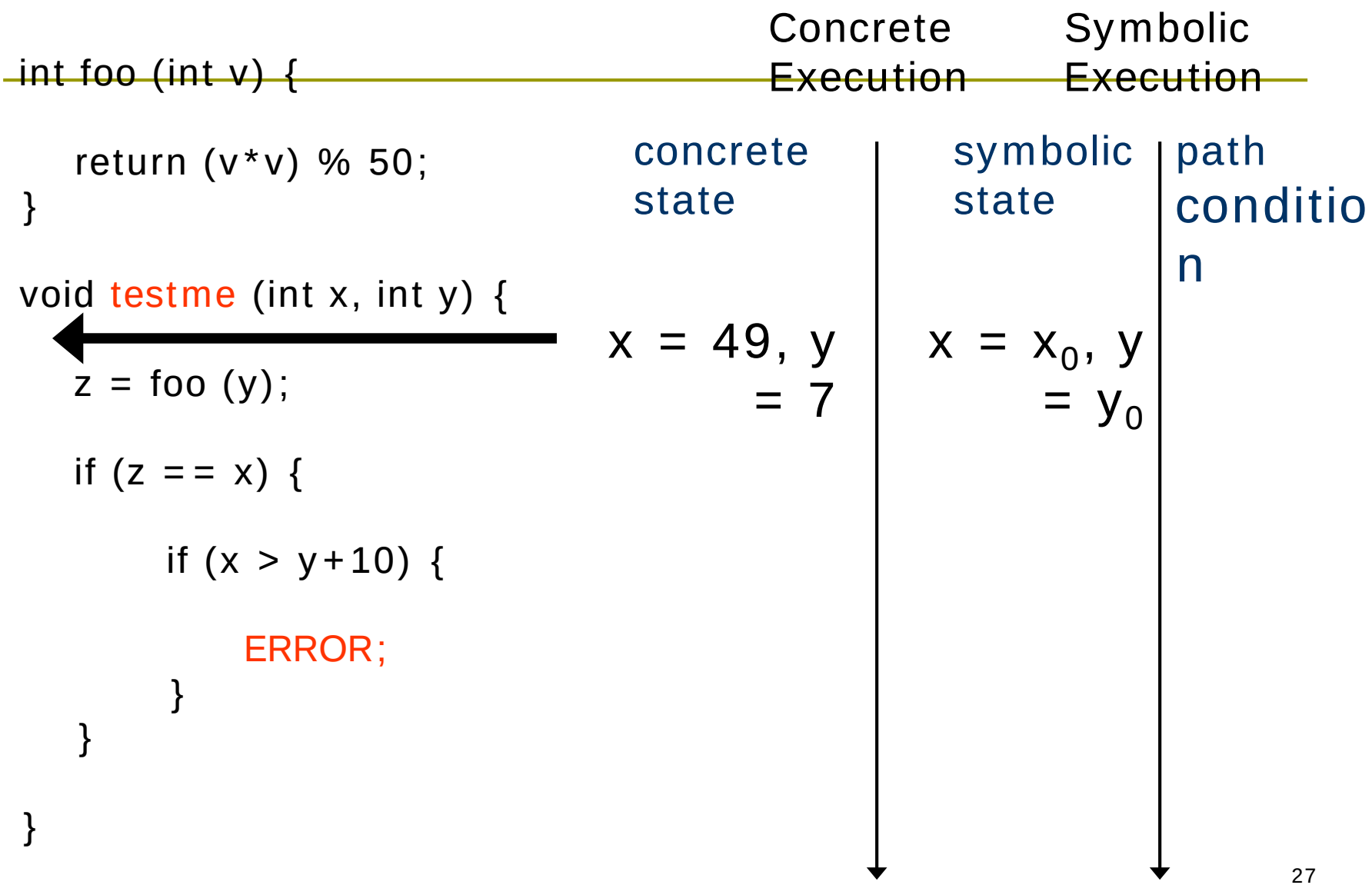
Novelty : Simultaneous Concrete and Symbolic Execution



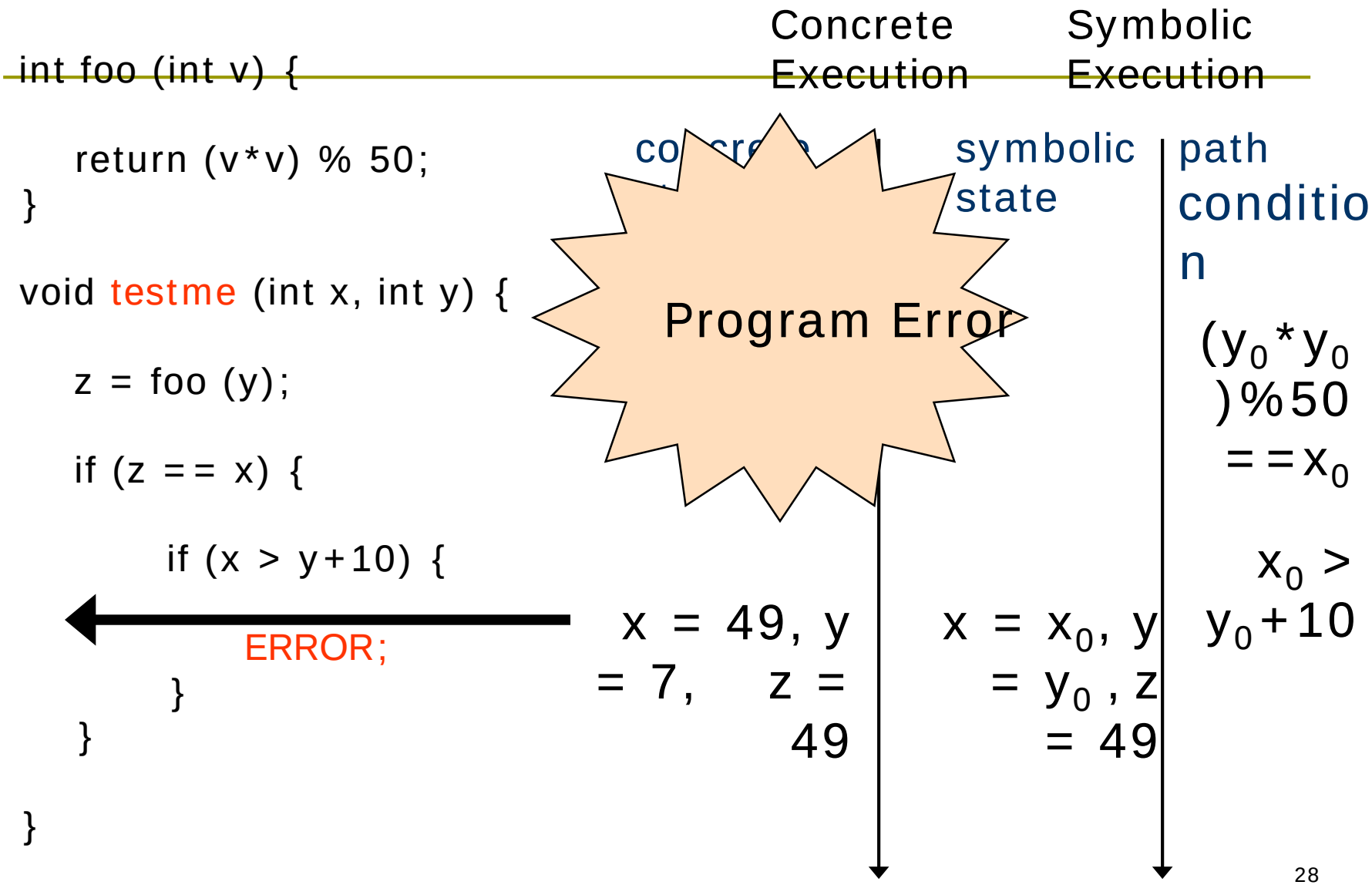
Novelty : Simultaneous Concrete and Symbolic Execution



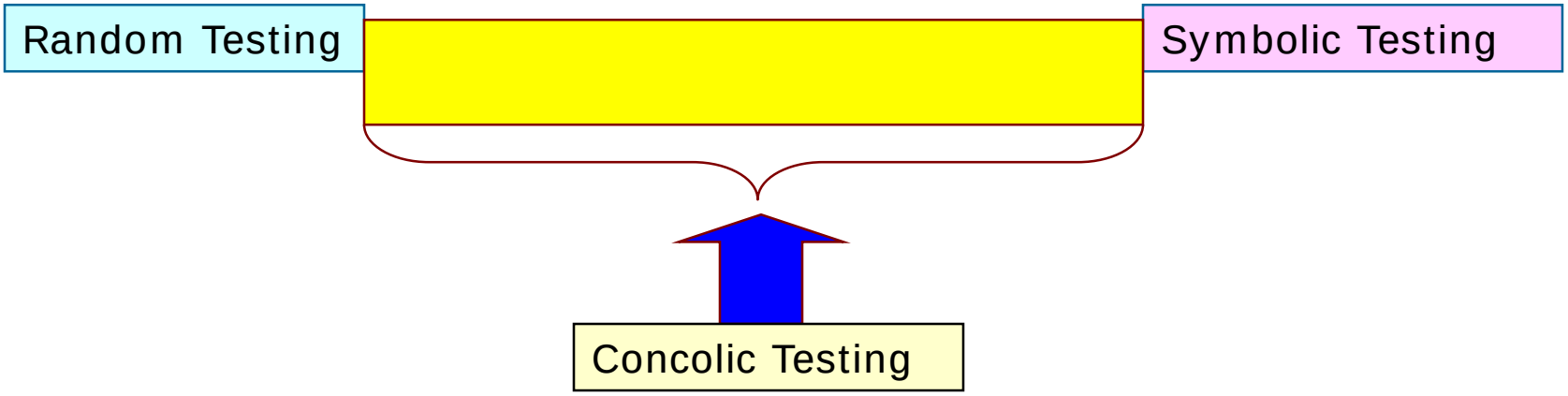
Novelty : Simultaneous Concrete and Symbolic Execution



Novelty : Simultaneous Concrete and Symbolic Execution



Concolic Testing: A Middle Approach



+ Complex programs
+ Efficient
- Less coverage
+ No false positive

+ Complex programs
+/- Somewhat efficient
+ High coverage
+ No false positive

- Simple programs
- Not efficient
+ High coverage
- False positive

Concolic Testing: Finding Security and Safety Bugs

Divide by 0 Error

```
x = 3 / i;
```

Buffer Overflow

```
a[i] = 4;
```

Concolic Testing: Finding Security and Safety Bugs

Key: Add Checks Automatically and Perform Concolic Testing

Divide by 0 Error

```
if (i !=0)
   x = 3 / i;
else
   ERROR;
```

Buffer Overflow

```
if (0<=i  &&  i <
   a.length)
   a[i] = 4;
else
   ERROR;
```

Implementations

- DART and CUTE for C programs
- jCUTE for Java programs
  - Goto http://srl.cs.berkeley.edu/~ksen/ for CUTE and jCUTE binaries
- MSR has four implementations
  - SAGE, PEX, YOGI, Vigilante
- Similar tool: EXE at Stanford
- Easiest way to use and to develop on top of CUTE
  - Implement concolic testing yourself

Pen and paper exercise

- Apply concolic execution on the following program

```
void hello(int x, int y) {
        int t = 0;
        if (x > y) {
                t = x * x - 3;
        } else {
                t = y;
        }

        if (t < x) {
                print("Hello World");
        }
}
```

Further reading

- Symbolic execution and program testing - James King
- KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs - Cadar et. al.
- Symbolic Execution for Software Testing: Three Decades Later - Cadar and Sen
- DART: Directed Automated Random Testing - Godefroid et. al.
- CUTE: A Concolic Unit Testing Engine for C - Sen et. al.