Symbolic and Concolic Testing

Nick Feng

Based on slides by Azadeh Farzan, Caroline Hu, Marsha Chechik and Michael Hicks

Symbolic Execution Summary

- A static analysis technique
- Symbolic values instead of concrete inputs.
- At each program location, the state is defined by:
  - current assignments to symbolic values and local variables.
  - a path condition that must hold for the execution to reach that location (conditions on the inputs to reach the location).

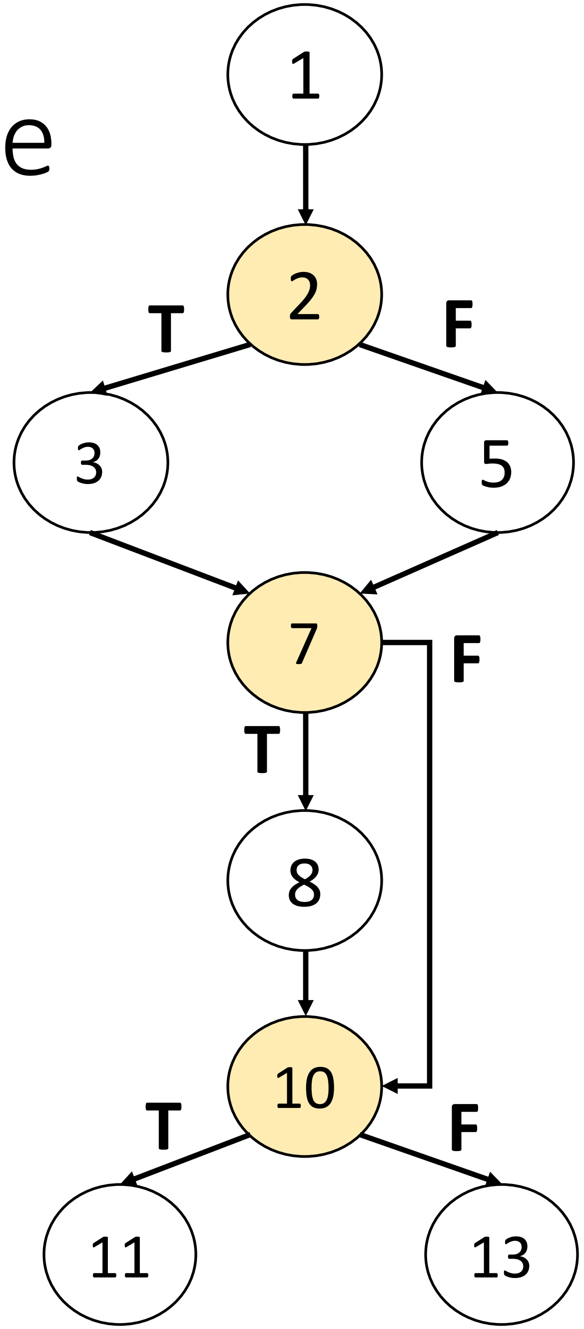
Slide Credit: Azadeh Farzan

Symbolic Execution Summary

- At each branch, both paths are followed.
  - On the true branch: the condition is added to the path constraints.
  - On the false branch: the negation of the condition is added.
- If the branch is infeasible, execution stops.

Slide Credit: Azadeh Farzan

Example

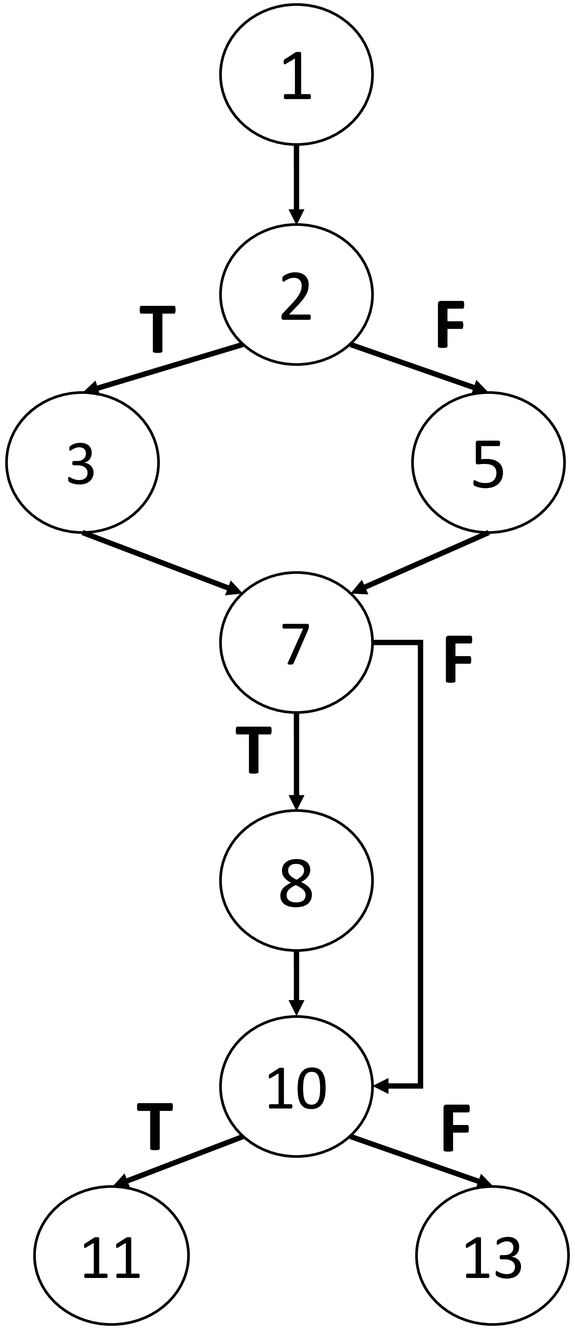


Branching Conditions

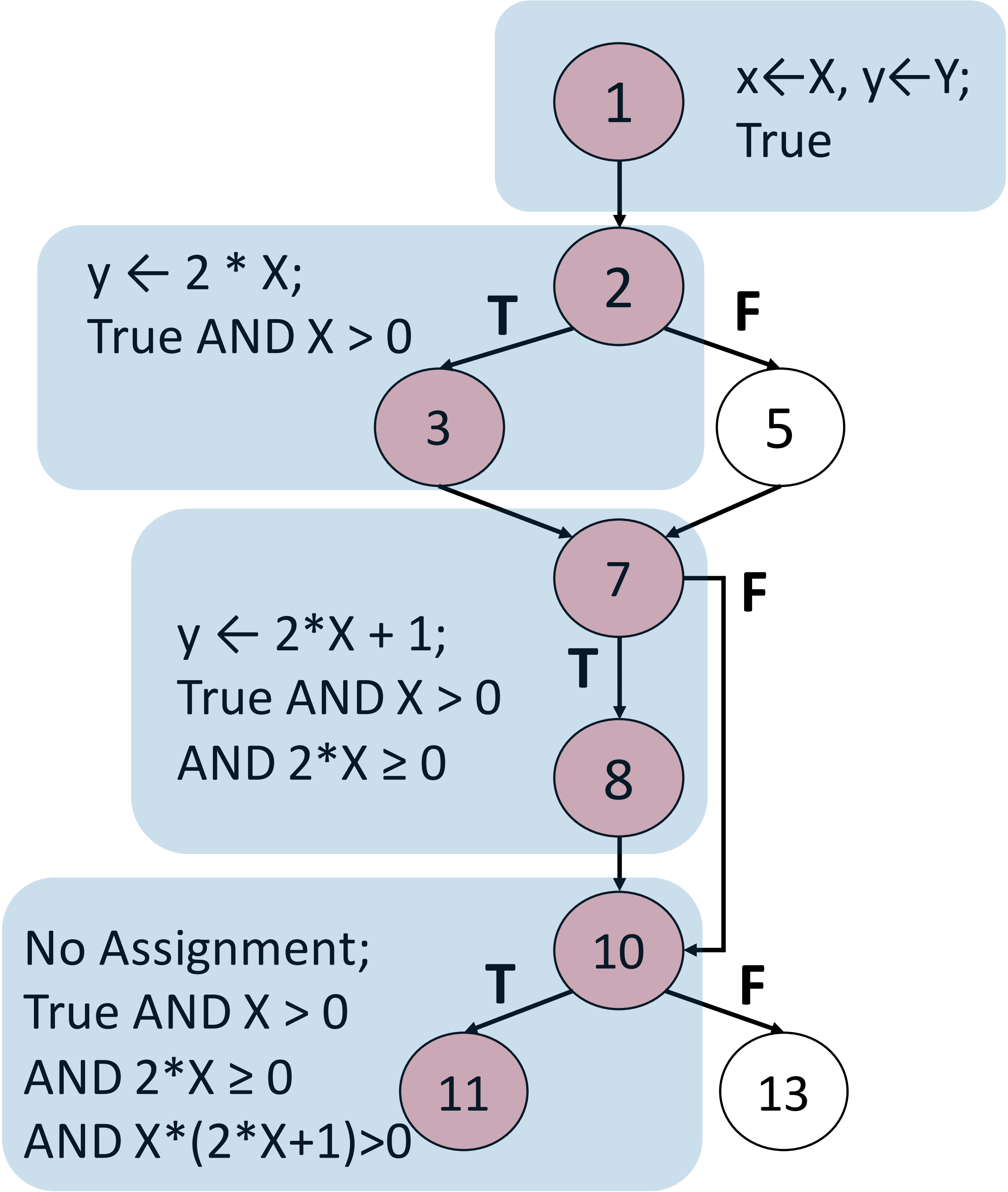
```
Function foo (int x, int y):
    Read x, y
    if x > 0:
        y = 2 * x
    else:
        y = x
    endif
    if y ≥ 0:
        y = y + 1
    endif
    if x * y > 0:
        return x
    else:
        return y
```

Example

- Input variables: x, y
  - Symbolic Names: X, Y
- All paths?
  - 1, 2, 3, 7, 8, 10, 11
  - 1, 2, 5, 7, 8, 10, 11
  - 1, 2, 3, 7, 10, 11
  - 1, 2, 5, 7, 10, 11
  - 1, 2, 3, 7, 8, 10, 13
  - 1, 2, 5, 7, 8, 10, 13
  - 1, 2, 3, 7, 10, 13
  - 1, 2, 5, 7, 10, 13




```
Function foo (int x, int y):
1       Read x, y
2       if x > 0:
3           y = 2 * x
4       else:
5           y = x
6       endif
7       if y ≥ 0:
8           y = y + 1
9       endif
10      if x * y > 0:
11          return x
12      else:
13          return y
```



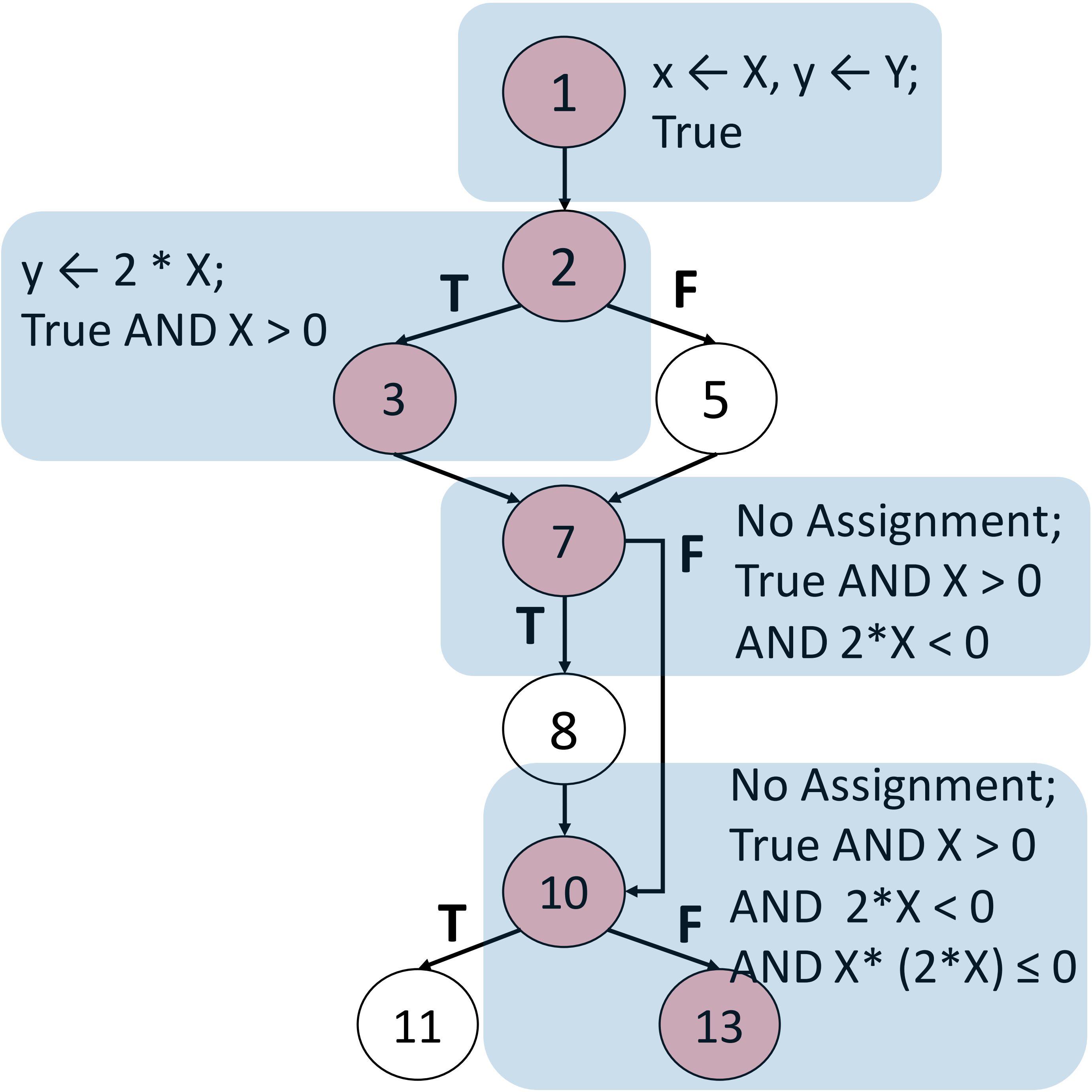

```
Function foo (int x, int y):
1     Read x, y
2     if x > 0:
3         y = 2 * x
4     else:
5         y = x
6     endif
7     if y ≥ 0:
8         y = y + 1
9     endif
10    if x * y > 0:
11        return x
12    else:
13        return y
```

Path: 1, 2, 3, 7, 8, 10, 11

line	Assignment	Path Condition
1	x ← X, y ← Y	True
2, 3	y ← 2 * X	True AND X > 0
7, 8	y ← 2*X + 1	True AND X > 0 AND 2*X ≥ 0
10, 11		True AND X > 0 AND 2*X ≥ 0 AND X * (2*X+1) > 0

Solvable: X =1, Y = 1, feasible path

```
Function foo (int x, int y):
1       Read x, y
2       if x > 0:
3           y = 2 * x
4       else:
5           y = x
6       endif
7       if y ≥ 0:
8           y = y + 1
9       endif
10      if x * y > 0:
11          return x
12      else:
13          return y
```




```
Function foo (int x, int y):
1      Read x, y
2      if x > 0:
3          y = 2 * x
4      else:
5          y = x
6      endif
7      if y ≥ 0:
8          y = y + 1
9      endif
10     if x * y > 0:
11         return x
12     else:
13         return y
```

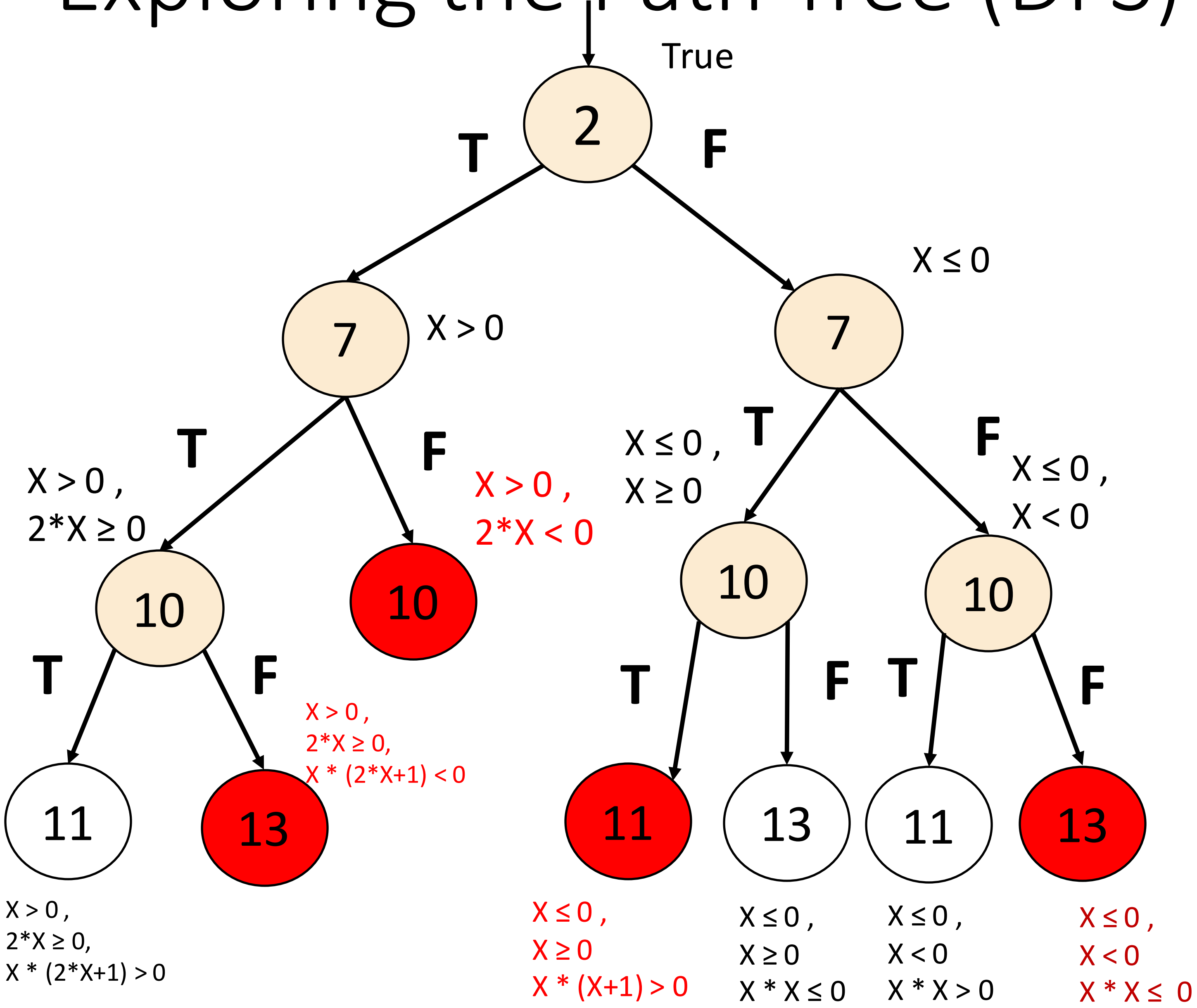
Path: 1, 2, 3, 7, 10, 13

line	Assignment	Path Condition
1	x ← X, y ← Y	True
2, 3	y ← 2 * X	True AND X > 0
7		True AND X > 0 AND 2*X < 0
10, 13		True AND X > 0 AND 2*X < 0 AND X* (2*X) ≤ 0

No Solution! Infeasible path

```
Function foo (int x, int y):
1       Read x, y
2       if x > 0:
3           y = 2 * x
4       else:
5           y = x
6       endif
7       if y ≥ 0:
8           y = y + 1
9       endif
10      if x * y > 0:
11          return x
12      else:
13          return y
```

Exploring the Path Tree (DFS)




```
Function foo (int x, int y):
    Read x, y
    if x > 0:
        y = 2 * x
    else:
        y = x
    endif
    if y ≥ 0:
        y = y + 1
    endif
    if x * y > 0:
        return x
    else:
        return y
```

Problem with Symbolic Execution

- Symbolic constraints can be very complex and cannot be solved by the constraint solver.
- The program being analyzed may have black box library functions.

Solution

Replace some of the symbolic values by concrete values available from the concrete state.

This is sound because concrete values are instantiations of symbolic values.

Slide credit: Marsha Chechik

Problem with Symbolic Execution and Solution

- However, the approach may lose completeness.
- Nevertheless, this way of replacing some symbolic values by concrete values helps concolic testing scale for large programs for which symbolic testing would have otherwise failed

Slide credit: Marsha Chechik

Concolic Testing

CONCRETE EXECUTION (random testing) +
SYMB**OLIC** EXECUTION (symbolic testing) =
CONCOLIC EXECUTION

```
int foo (int v):
      return (v*v) % 50

void testme (int x, int y):
      z = foo (y)
      if (z == x):
            if (x > y + 10):
                  Error
```

Slide credit: Marsha Chechik

Concolic Testing

```
int foo (int v):
    return (v*v) % 50

void testme (int x, int y):
    z = foo (y)
    if (z == x):
        if (x > y + 10):
            Error
```

Concrete Execution	Symbolic Execution	
Concrete State	Symbolic State	Path Condition
x = 22, y = 7	x = X, y = Y	

Slide credit: Marsha Chechik

Concolic Testing

```
   int foo (int v):
1       return (v*v) % 50

   void testme (int x, int y):
1       z = foo (y)
2       if (z == x):
3           if (x > y + 10):
4               Error
```

Concrete State	Symbolic State	Path Condition
x = 22, y = 7	x = X, y = Y	
x = 22, y = 7, z = 49	x = X, y = Y, z = (Y * Y) % 50	

Concrete Execution | Symbolic Execution

Slide credit: Marsha Chechik

Concolic Testing

```
   int foo (int v):
1       return (v*v) % 50

   void testme (int x, int y):
1       z = foo (y)
2       if (z == x):
3           if (x > y + 10):
4               Error
```

Concrete Execution	Symbolic Execution	
Concrete State	**Symbolic State**	**Path Condition**
x = 22, y = 7	x = X, y = Y	
x = 22, y = 7, z = 49	x = X, y = Y, z = (Y * Y) % 50	
		(Y * Y) % 50 != X

Slide credit: Marsha Chechik

Concolic Testing

```
int foo (int v):
    return (v*v) % 50

void testme (int x, int y):
    z = foo (y)
    if (z == x):
        if (x > y + 10):
            Error
```

Concrete Execution | Symbolic Execution

Concrete State	Symbolic State	Path Condition
		(Y * Y) % 50 != X

Solve: (Y*Y) % 50 = X

Solution ?

Slide credit: Marsha Chechik

Concolic Testing

- Deals with black box library functions
  - Replace symbolic values by concrete values
  - Ex.

```
int foo (int v):
        return v*v % 50
void testme (int x, int y):
        z = foo (y)
        if (z == x):
                if (x > y + 10):
                        Error
```

Solve for Y * Y % 50 = X

Solution: X = 49, Y = 7

- Deals with complex symbolic constraints: (Y*Y) % 50 = X

Concolic Testing

```
   int foo (int v):
1       return (v*v) % 50

   void testme (int x, int y):
1       z = foo (y)
2       if (z == x):
3           if (x > y + 10):
4               Error
```

Concrete Execution	Symbolic Execution	
Concrete State	Symbolic State	Path Condition

Solve: (Y*Y) % 50 = X

Solution ?

When the constraint is complex, use concrete state

Replace Y by 7

(Y * Y) % 50 != X

Slide credit: Marsha Chechik

Concolic Testing

```
int foo (int v):
    return (v*v) % 50

void testme (int x, int y):
    z = foo (y)
    if (z == x):
        if (x > y + 10):
            Error
```

Concrete State	Symbolic State	Path Condition
		(Y * Y) % 50 != X

Concrete Execution

Symbolic Execution

Solution: X = 49, Y = 7

Slide credit: Marsha Chechik

Concolic Testing

```
int foo (int v):
    return (v*v) % 50

void testme (int x, int y):
    z = foo (y)
    if (z == x):
        if (x > y + 10):
            Error
```

Concrete Execution	Symbolic Execution	
Concrete State	Symbolic State	Path Condition
x = 49, y = 7	x = X, y = Y	

Slide credit: Marsha Chechik

Concolic Testing

```
int foo (int v):
    return (v*v) % 50

void testme (int x, int y):
    z = foo (y)
    if (z == x):
        if (x > y + 10):
            Error
```

Concrete Execution	Symbolic Execution	
Concrete State	Symbolic State	Path Condition
x = 49, y = 7	x = X, y = Y	
x = 49, y = 7, z = 49	x = X, y = Y, z = 49	
		Z == X X > Y + 10

Slide credit: Marsha Chechik

Concolic Testing

Concrete Execution | Symbolic Execution

```
int foo (int v):
    return (v*v) % 50

void testme (int x, int y):
    z = foo (y)
    if (z == x):
        if (x > y + 10):
            Error
```

Concrete State	Symbolic State	Path Condition
x = 49, = 7	= X, y = Y	
	y = Y,	
		Z == X
		X > Y + 10

Program Error

Slide credit: Marsha Chechik

Concolic Testing

```
int foo (int v):
    return (v*v) % 50

void testme (int x, int y):
    z = foo (y)
    if (z == x):
        if (z < 10):
            Error
```

Concrete Execution	Symbolic Execution	
Concrete State	Symbolic State	Path Condition
x = 49, y = 7	x = X, y = Y	
x = 49, y = 7, z = 49	x = X, y = Y, z = 49	
		Z == X Z ≥ 10

Slide credit: Marsha Chechik

Concolic Testing

```
int foo (int v):
    return (v*v) % 50

void testme (int x, int y):
    z = foo (y)
    if (z == x):
        if (z < 10):
            Error
```

Concrete Execution	Symbolic Execution	
Concrete State	Symbolic State	Path Condition
x = 49, y = 7	x = X, y = Y	
x = 49, y = 7,	x = X, y = Y,	
		Z == X
		Z ≥ 10

Solve: 49 == X ,
49 < 10

No Solution!

Slide credit: Marsha Chechik

Concolic Testing

```
int foo (int v):
    return (v*v) % 50

void testme (int x, int y):
    z = foo (y)
    if (z == x):
        if (z < 10):
            Error
```

Concrete State	Symbolic State	Path Condition
x = 1, y = 1	= X, y = Y	
	y = Y,	
		Z = X Z < 10

Concrete Execution — Symbolic Execution

Concretization may lose Completeness

The Path is feasible with X= 1, Y =1

Slide credit: Marsha Chechik

Concolic Testing Example

Concrete Input: X = 0, Y = 0

Explored path:

```
Function foo (int x, int y):
    Read x, y
    if x > 0:
        y = 2 * x
    else:
        y = x
    endif
    if y ≥ 0:
        y = y + 1
    endif
    if x * y > 0:
        return x
    else:
        return y
```

Concolic Testing Example

Concrete Input: X = 0, Y = 0
Execution Path: FTF

Explored path:
FTF

Unexplored Path's PC:

1. $X > 0$
2. $X \leq 0 \wedge X < 0$
3. $X \leq 0 \wedge X \geq 0 \wedge X * (X + 1) > 0$

Now, pick one path to explore!

```
Function foo (int x, int y):
    Read x, y
    if x > 0:
        y = 2 * x
    else:
        y = x
    endif
    if y ≥ 0:
        y = y + 1
    endif
    if x * y > 0:
        return x
    else:
        return y
```

Concolic Testing Example

Concrete Input: X = 0, Y = 0
Execution Path: FTF

Explored path:
FTF

Unexplored Path's PC:

1. $X > 0$
2. $X \leq 0 \wedge X < 0$
3. $X \leq 0 \wedge X \geq 0 \wedge X * (X + 1) > 0$

Pick the first PC, solving yields:
X = 1, Y = 0

```
Function foo (int x, int y):
    Read x, y
    if x > 0:
        y = 2 * x
    else:
        y = x
    endif
    if y ≥ 0:
        y = y + 1
    endif
    if x * y > 0:
        return x
    else:
        return y
```

Concolic Testing Example

Concrete Input: X = 1, Y = 0
Execution Path: TTT

Explored path:
FTF
TTT

Unexplored Path's PC:

1. $X \leq 0 \wedge X < 0$
2. $X \leq 0 \wedge X \geq 0 \wedge X * (X + 1) > 0$
3. $X > 0 \wedge 2X < 0$
4. $X > 0 \wedge 2X \geq 0 \wedge (2X + 1) \leq 0$

Pick the first PC, solving yields:
X = -1, Y = 0

```
Function foo (int x, int y):
    Read x, y
    if x > 0:
        y = 2 * x
    else:
        y = x
    endif
    if y ≥ 0:
        y = y + 1
    endif
    if x * y > 0:
        return x
    else:
        return y
```

Concolic Testing Example

Concrete Input: X = -1, Y = 0
Execution Path: FFT

Explored path:
FTF
TTT
FFT

Unexplored Path's PC:

1. $X \leq 0 \land X \geq 0 \land X * (X+1) > 0$
2. $X > 0 \land 2X < 0$
3. $X > 0 \land 2X \geq 0 \land (2X+1) \leq 0$
4. $X \leq 0 \land X < 0 \land X * X \leq 0$

Pick the first PC, but it has no solution, remove it

```
Function foo (int x, int y):
    Read x, y
    if x > 0:
        y = 2 * x
    else:
        y = x
    endif
    if y ≥ 0:
        y = y + 1
    endif
    if x * y > 0:
        return x
    else:
        return y
```

Concolic Testing Example

Concrete Input: X = -1, Y = 0
Execution Path: FFT

Explored path:
FTF
TTT
FFT

Unexplored Path's PC:

1. $X > 0 \wedge 2X < 0$
2. $X > 0 \wedge 2X \geq 0 \wedge (2X + 1) \leq 0$
3. $X \leq 0 \wedge X < 0 \wedge X * X \leq 0$

Pick the first PC, but it has no solution, remove it

```
Function foo (int x, int y):
    Read x, y
    if x > 0:
        y = 2 * x
    else:
        y = x
    endif
    if y ≥ 0:
        y = y + 1
    endif
    if x * y > 0:
        return x
    else:
        return y
```

Concolic Testing Example

Concrete Input: X = -1, Y = 0

Execution Path: FFT

Unexplored Path's PC:

1. $X > 0 \wedge 2X \geq 0 \wedge (2X + 1) \leq 0$
2. $X \leq 0 \wedge X < 0 \wedge \ X * X \leq 0$

Explored path:

FTF

TTT

FFT

Pick the first PC, but it has no solution, remove it

```
Function foo (int x, int y):
    Read x, y
    if x > 0:
        y = 2 * x
    else:
        y = x
    endif
    if y ≥ 0:
        y = y + 1
    endif
    if x * y > 0:
        return x
    else:
        return y
```

Concolic Testing Example

Concrete Input: X = -1, Y = 0
Execution Path: FFT

Unexplored Path's PC:

1. $X \leq 0 \wedge X < 0 \wedge X * X \leq 0$

Explored path:
FTF
TTT
FFT

Pick the first PC, but it has no solution, remove it

```
Function foo (int x, int y):
    Read x, y
    if x > 0:
        y = 2 * x
    else:
        y = x
    endif
    if y ≥ 0:
        y = y + 1
    endif
    if x * y > 0:
        return x
    else:
        return y
```

Concolic Testing Example

Concrete Input: X = -1, Y = 0
Execution Path: FFT

Unexplored Path's PC:

Explored path:
FTF
TTT
FFT

No more unexplored path, we are done!

```
Function foo (int x, int y):
    Read x, y
    if x > 0:
        y = 2 * x
    else:
        y = x
    endif
    if y ≥ 0:
        y = y + 1
    endif
    if x * y > 0:
        return x
    else:
        return y
```

The Path Explosion Problem

- The program under test may have too many paths
  - Number of paths is exponential in branching structure
  - Infinitely many paths for programs with unbound loop

```
int main() {
    int p, n, b;
    p = 42;
    if (b == 1 && n < 500  ){
        Possible Error
    }
    while ( n>=0 ) {
        assert p != 0;
        if (n == 0) {
            p = 0;
        }
        n--;
    }
    return 0;
}
```

- We want to find as many bugs as possible with limited resources

Search Strategy

- Basic search: BFS, DFS
- Random search
- Coverage guided heuristic
- Generation search
- Fuzzing + Symbolic Execution

Slide credit: Michael Hicks

Basic Search: BFS, DFS

- DFS: Depth-First Search:
  - Pick an unexplored direction from the last encountered branch point
  - The search order we followed for concolic testing example! (Slides 27-34)
- BFS: Breath-First Search:
  - Pick an unexplored direction from the first encountered branch point
  - The search order we followed for exploring the path Tree (Slides 10)

- Neither is guided by program knowledge
- DFS can get stuck on a part of the program

Slide credit: Michael Hicks

We can do better if we know where the possible error locations are

```
int main() {
    int p, n, b;
    p = 42;
    if (b == 1 & n == 500  ){
        Possible Error
    }
    while ( n>=0 ) {
      assert p != 0;
      if (n == 0) {
          p = 0;
      }
      n--;
    }
    return 0;
}
```

If we start with b=0, DFS could stuck in exploring the loop

Random Search

- If we don't know a priori which paths to take, then adding randomness seems like a good idea:
  - Idea 1: pick the next path to explore uniformly random
  - Idea 2: randomly restart search if haven't anything interesting
  - Idea 3: when have equal priority paths to explore, choose the next one randomly
- Drawback: reproducibility
  - Use pseudo-random with fixed seed instead

Slide credit: Michael Hicks

Coverage guided heuristic

- Idea: Try to visit statement we haven't seen before
- Approach:
  - Score of statement: statement visited count and frequency
  - Pick next statement with lowest score (more likely to discover new behavior)
- Why might this work?
  - Error are often in hard-to-reach parts of the program
  - This strategy tries to reach everywhere
- Why might this not work?
  - How do we get to a statement if proper precondition is not set up

Slide credit: Michael Hicks

Coverage guided heuristic

```
int main() {
    int p, n, b;
    p = 42;
    if (b == 1 & n == 500  ){
        Possible Error
    }
    while ( n>=0 ) {
        assert p != 0;
        if (n == 0) {
            p = 0;
        }
        n--;
    }
    return 0;
}
```

If we start with b=0, and stuck on exploring the loop

High score for statements inside the loop

Low score for Possible error

Prioritize the path with PC: b ==1 & n==500

Generational search

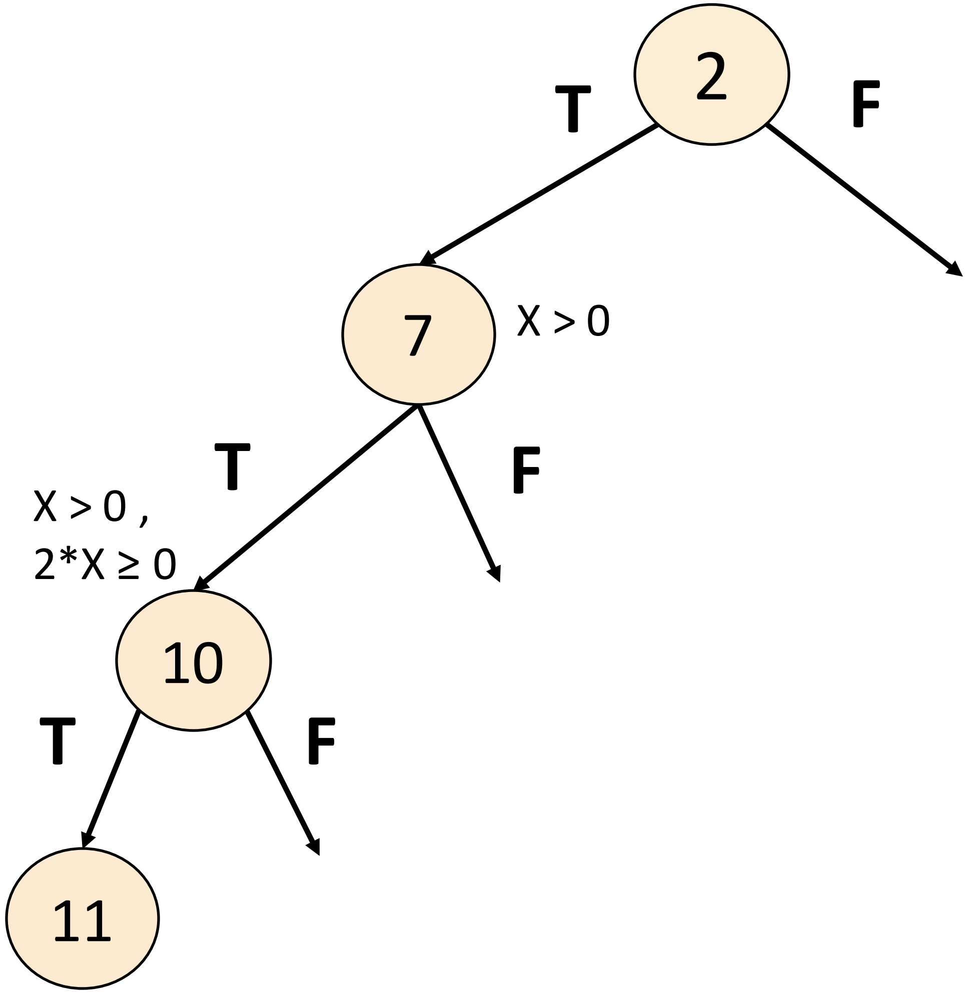
- Hybrid of BFS and coverage-guided
- Generation 0: a complete run of the program
- Generation 1: paths produced by negating one of the branch conditions from the PC of generation 0.
- Generation n: similar, but branching off from gen n-1
- Use a coverage heuristic (maximize block coverage) to pick priority

- The choice of Generation 0 is important! This could be an effective strategy with some prior knowledge about the program

Godefroid, Patrice, Michael Y. Levin, and David A. Molnar. "Automated Whitebox Fuzz Testing." *NDSS*. Vol. 8. 2008.

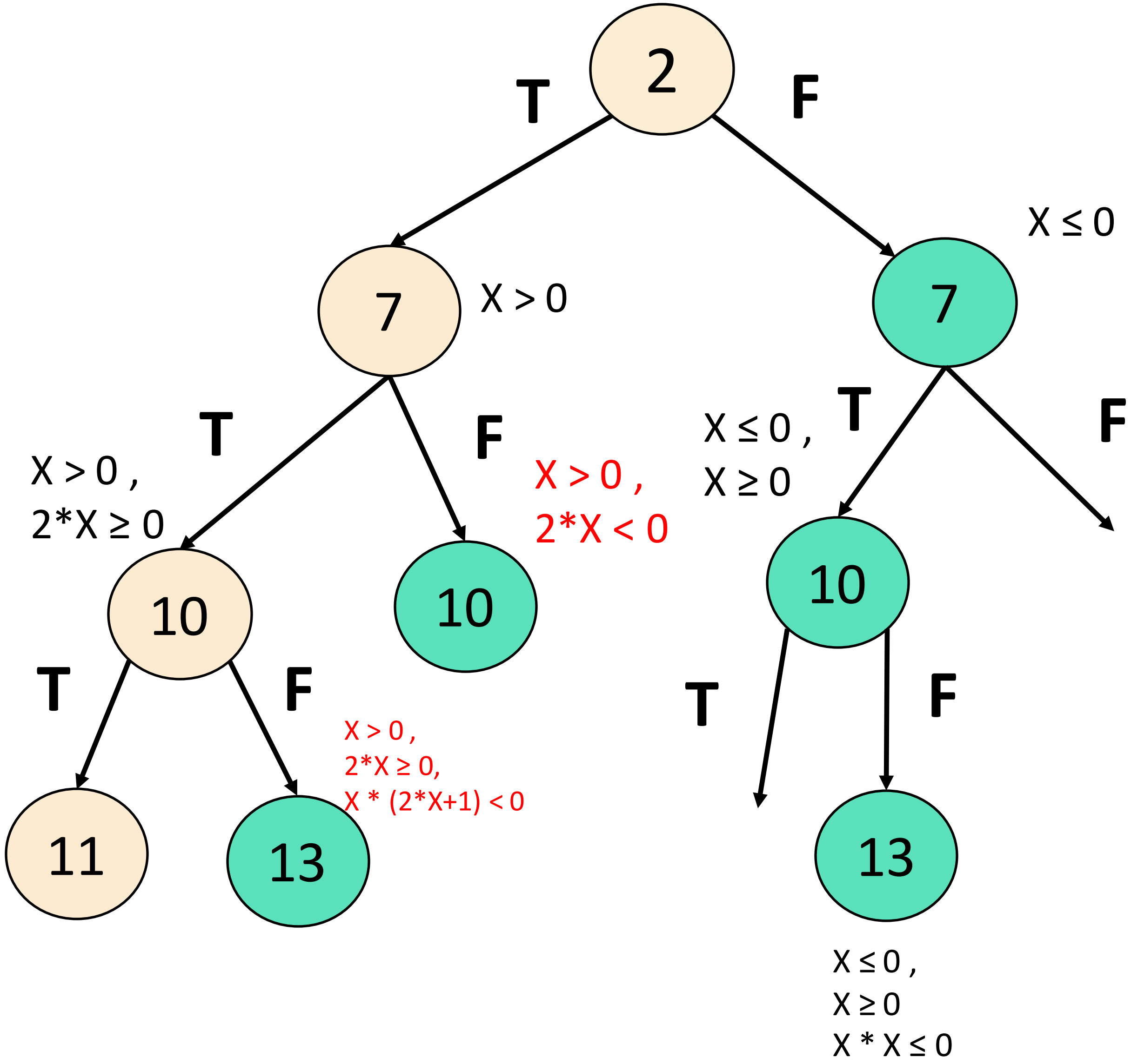
Slide credit: Michael Hicks

Generational search



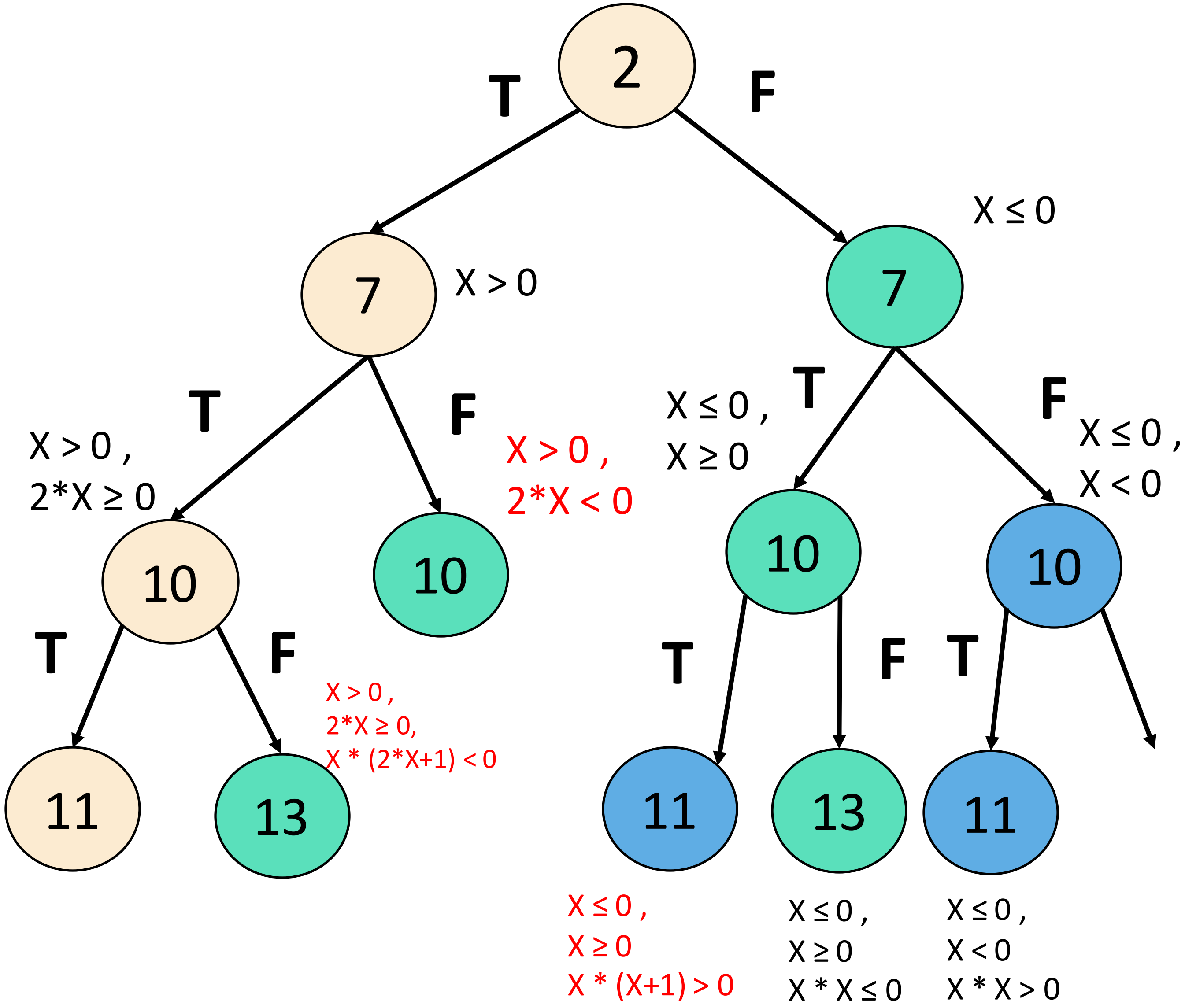
Gen 0:
TTT

Generational search



Gen 0:
TTT
Gen 1:
TF
TTF
FTF

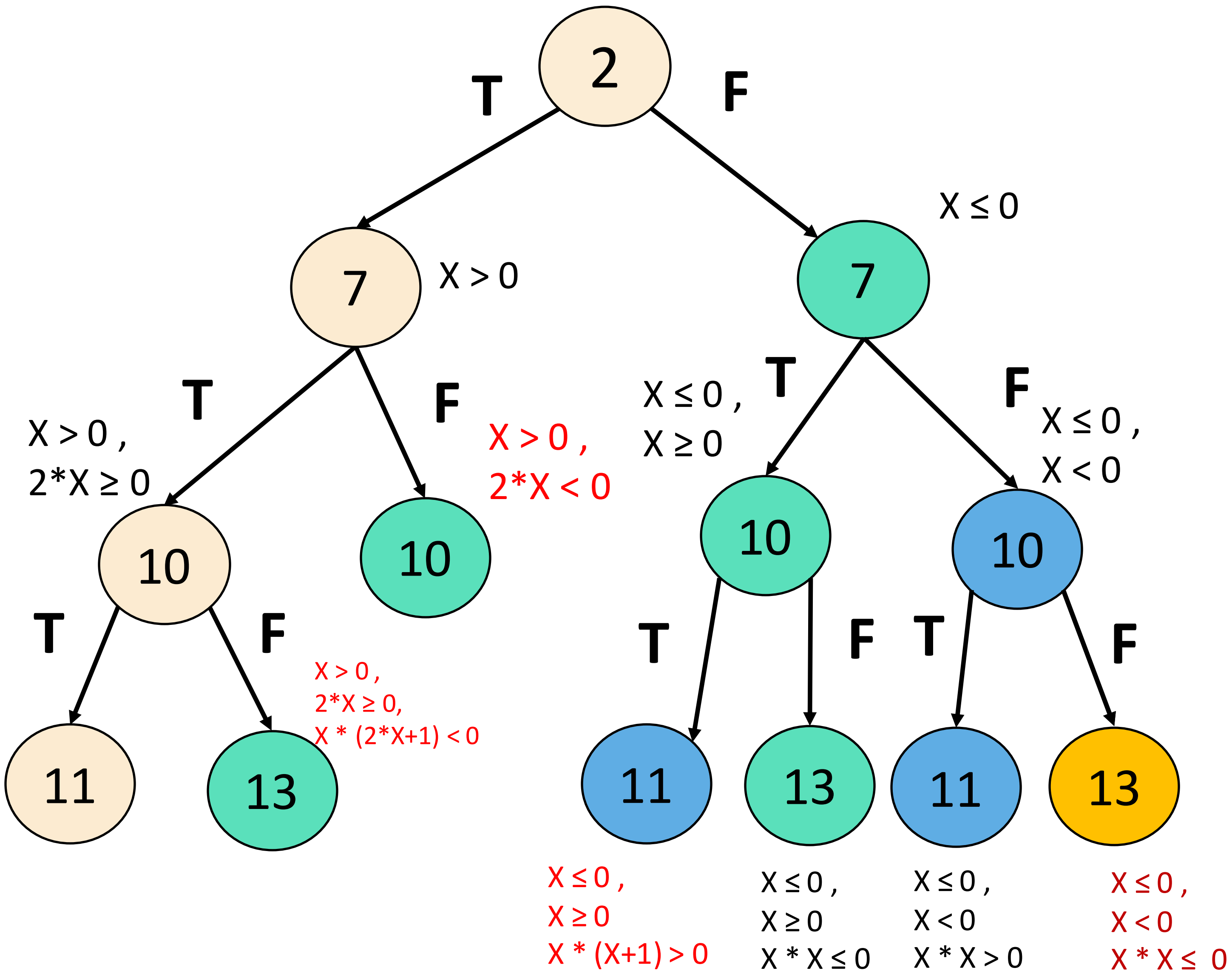
Generational search



Gen 0:
TTT
Gen 1:
TF
TTF
FTF

Gen 2
FTT
FFT

Generational search



Gen 0:
TTT

Gen 1:
TF
TTF
FTF

Gen 2
FTT
FFT

Gen 3
FFF

Random Testing (Fuzzing) + Symbolic Execution

- Fuzzing is simple, cheap and effective in achieving code coverage and finding shallow bugs.
- But cannot effectively explore paths and detect path-specific deep bugs

- Symbolic execution is effective at explore path specific deep bugs
- But expensive

- Combine both:
  - Use symbolic execution to discover preconditioned inputs as new seed for fuzzing
  - alternate between fuzzing and symbolic execution. Switch to symbolic execution to explore unexplored paths when fuzzing becomes less effective.

Pak, Brian S. "Hybrid fuzz testing: Discovering software bugs via fuzzing and symbolic execution." *School of Computer Science Carnegie Mellon University* (2012).