

Symbolic Execution and Concolic Testing

CSC410 Tutorial

Oct 9, 2026

Ruotong Cheng

Symbolic Execution

- A static analysis technique
- Usage
 - Compute summary
 - Find the reachable program locations
 - Find bugs
 - ...

Symbolic Execution: Steps

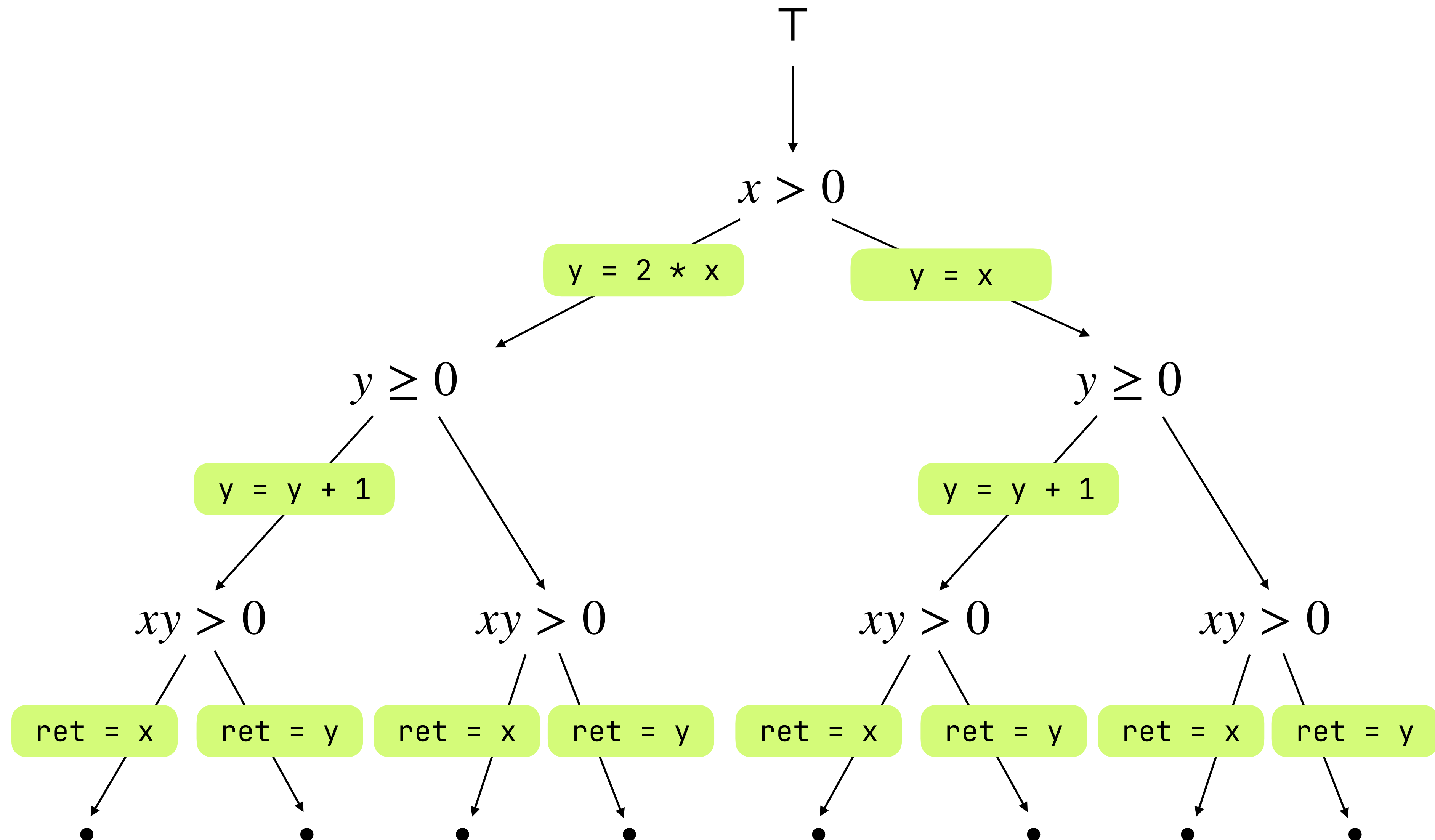
1. Draw the computation tree
2. Compute the symbolic states
3. Cross out the infeasible paths

Symbolic Execution: Example

```
def foo(x, y):  
    if x > 0:  
        y = 2 * x  
    else:  
        y = x  
    if y ≥ 0:  
        y = y + 1  
    if x * y > 0:  
        return x  
    else:  
        return y
```

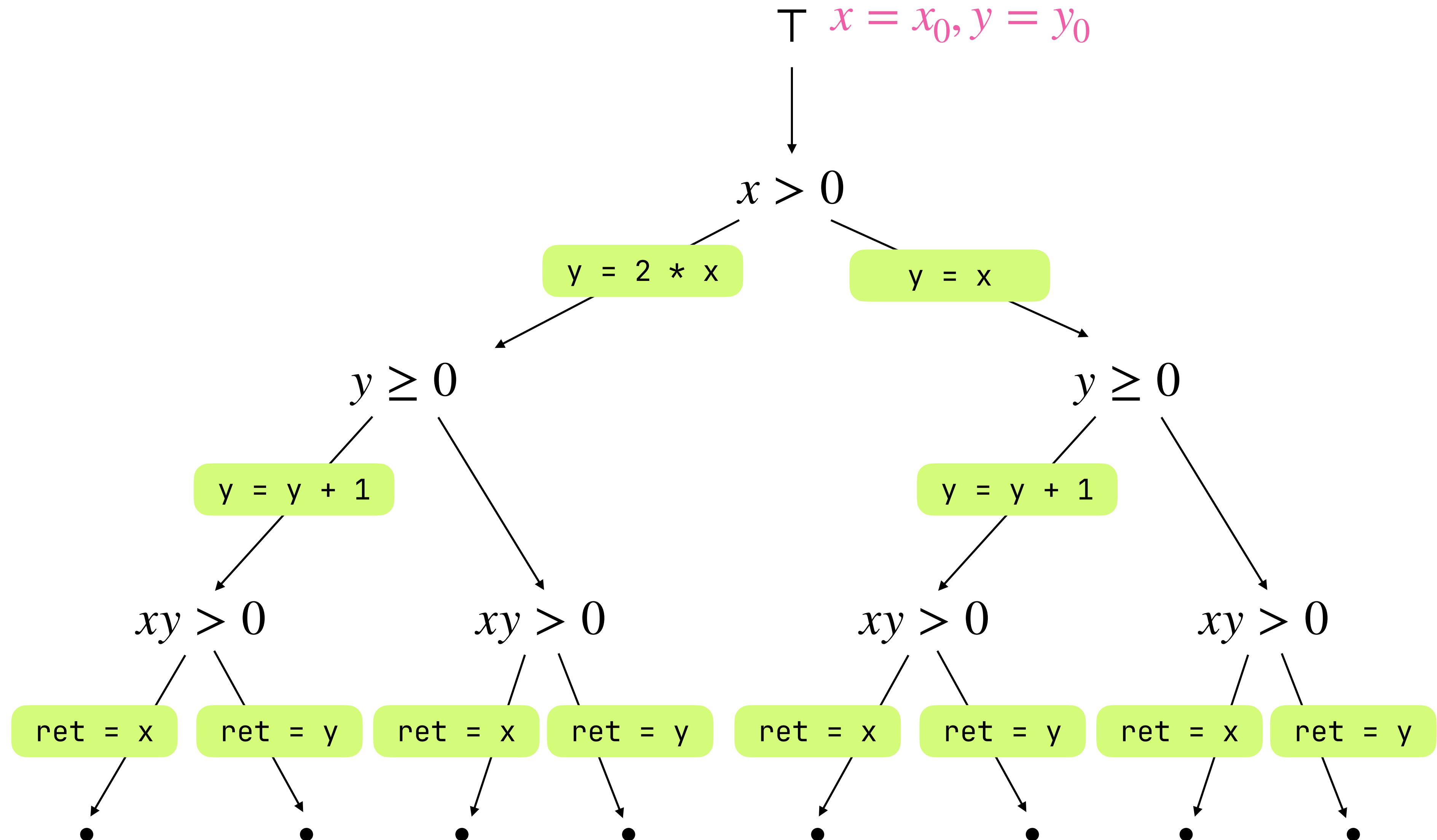
Symbolic Execution: Example

```
def foo(x, y):  
    if x > 0:  
        y = 2 * x  
    else:  
        y = x  
    if y ≥ 0:  
        y = y + 1  
    if x * y > 0:  
        return x  
    else:  
        return y
```



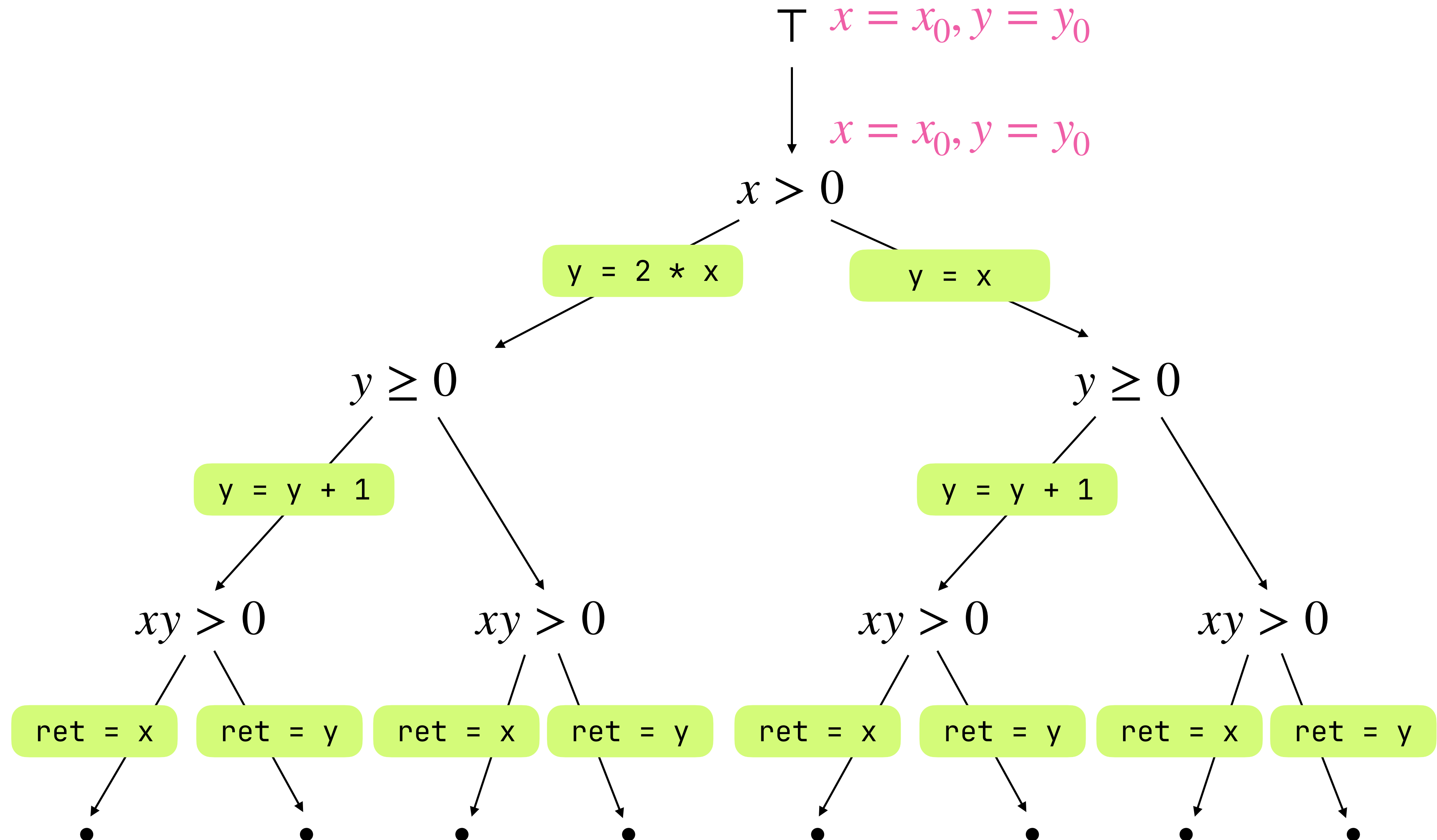
Symbolic Execution: Example

```
def foo(x, y):  
    if x > 0:  
        y = 2 * x  
    else:  
        y = x  
    if y ≥ 0:  
        y = y + 1  
    if x * y > 0:  
        return x  
    else:  
        return y
```



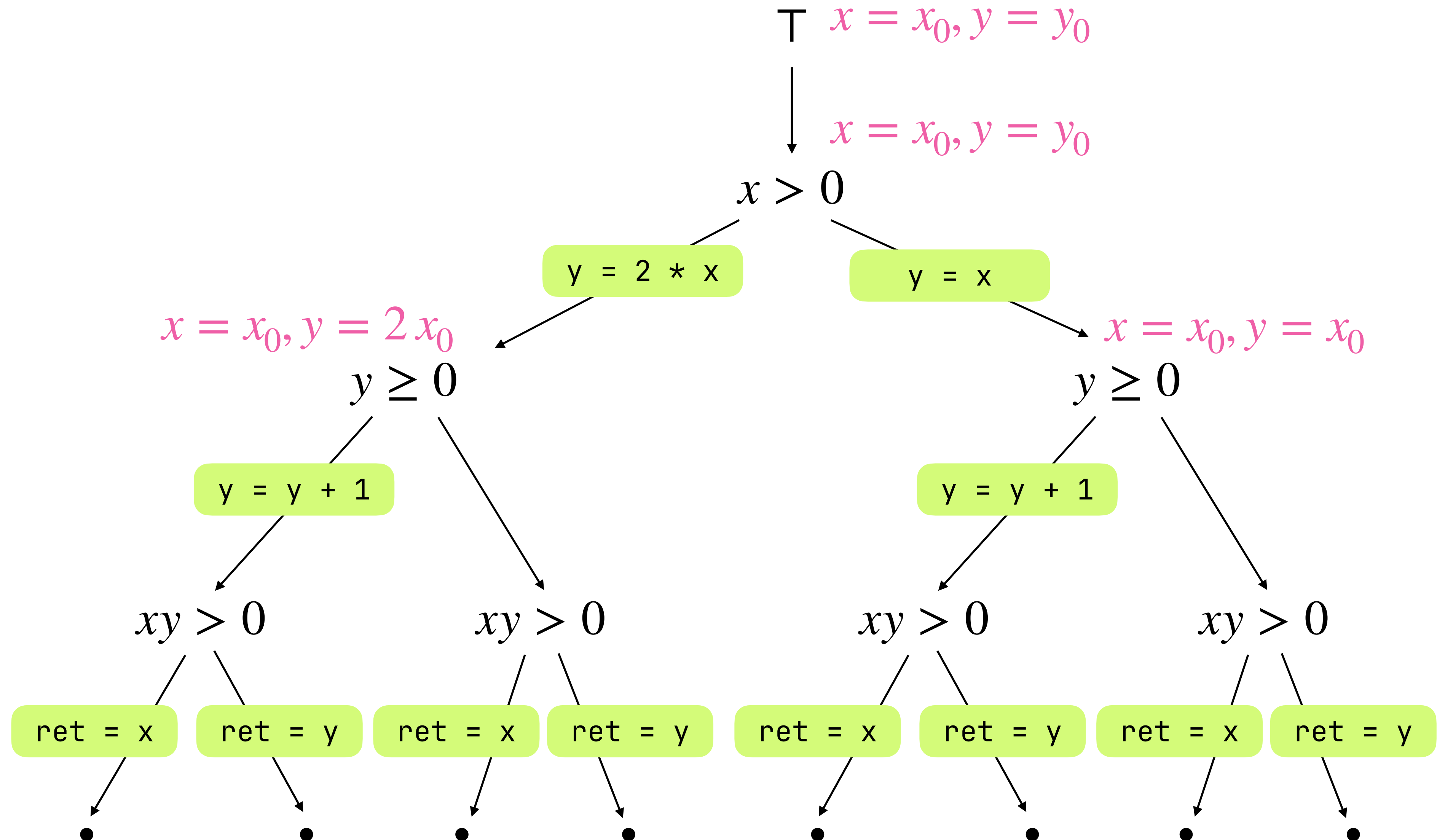
Symbolic Execution: Example

```
def foo(x, y):  
    if x > 0:  
        y = 2 * x  
    else:  
        y = x  
    if y ≥ 0:  
        y = y + 1  
    if x * y > 0:  
        return x  
    else:  
        return y
```



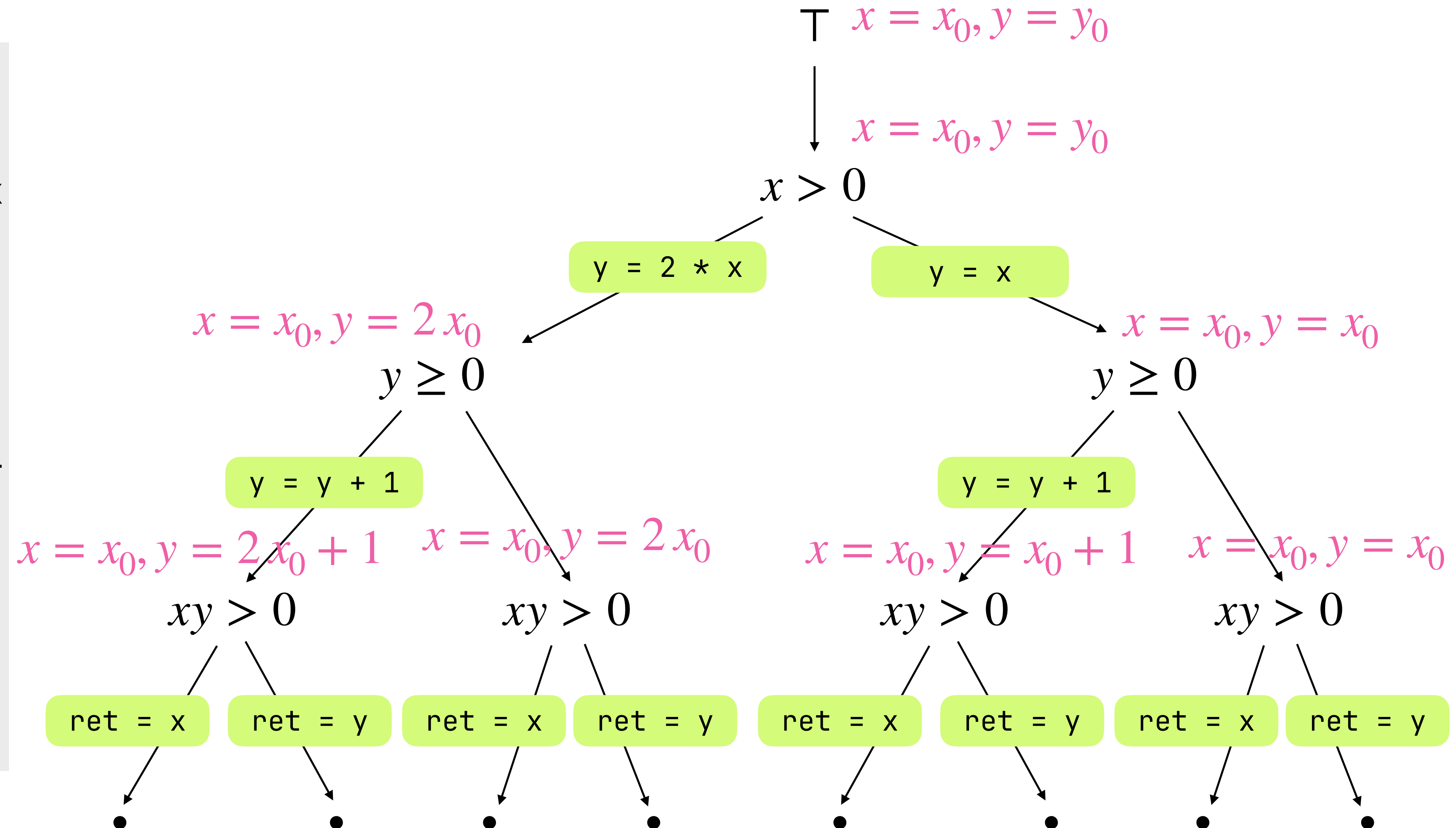
Symbolic Execution: Example

```
def foo(x, y):  
    if x > 0:  
        y = 2 * x  
    else:  
        y = x  
        if y ≥ 0:  
            y = y + 1  
        if x * y > 0:  
            return x  
    else  
        return y
```

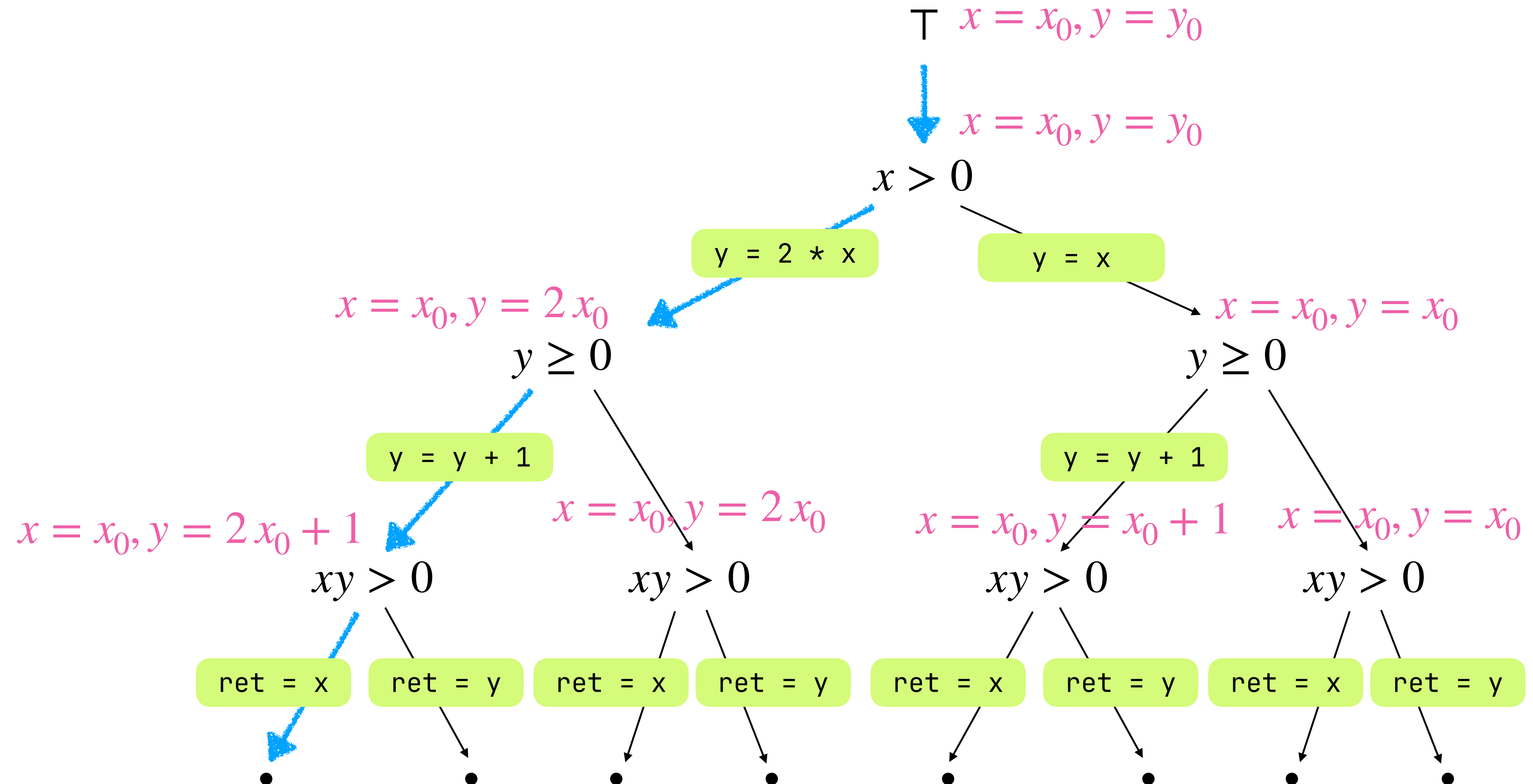


Symbolic Execution: Example

```
def foo(x, y):  
    if x > 0:  
        y = 2 * x  
    else:  
        y = x  
    if y ≥ 0:  
        y = y + 1  
    if x * y > 0:  
        return x  
    else:  
        return y
```



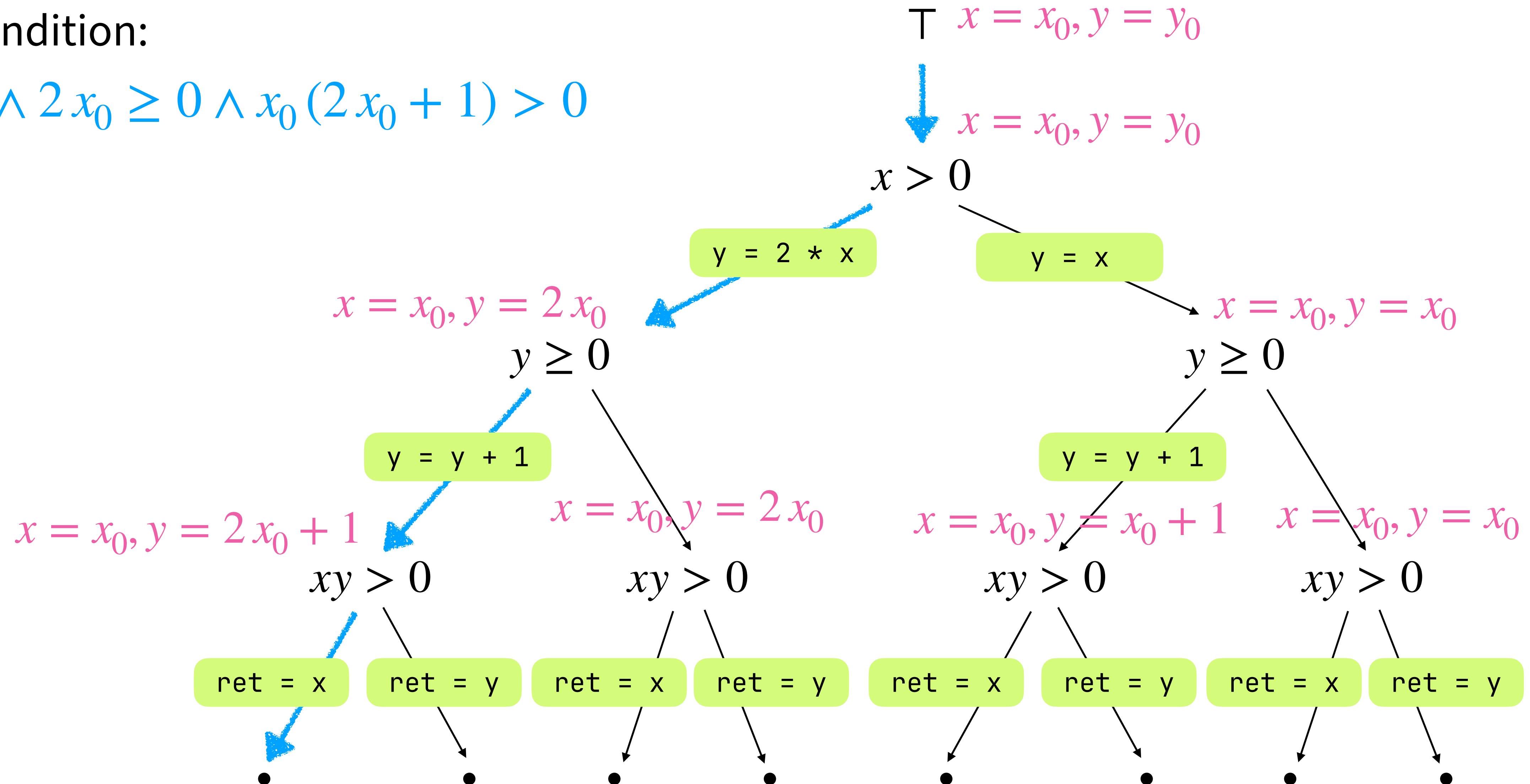
Symbolic Execution: Example



Symbolic Execution: Example

Path Condition:

$$x_0 > 0 \wedge 2x_0 \geq 0 \wedge x_0(2x_0 + 1) > 0$$

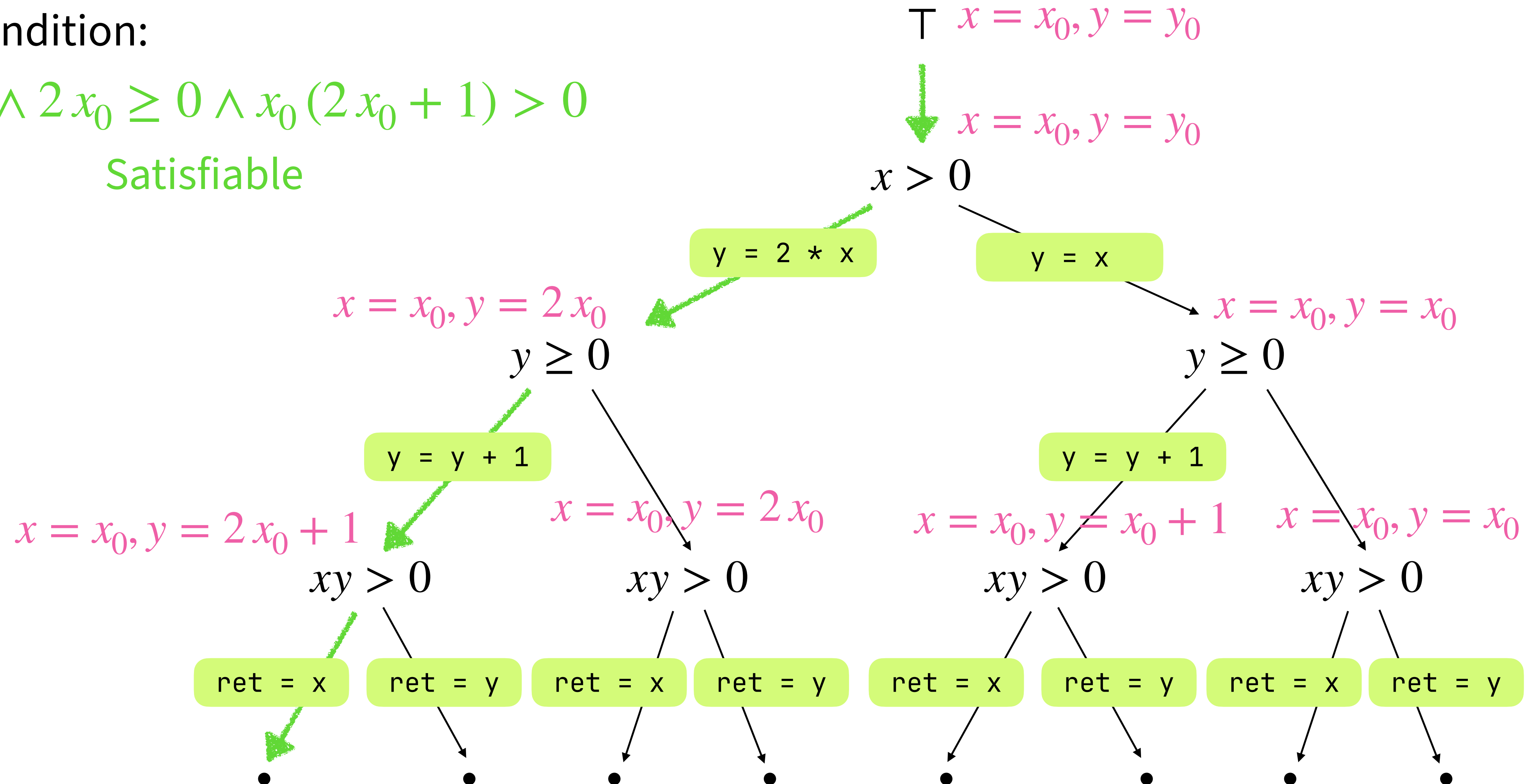


Symbolic Execution: Example

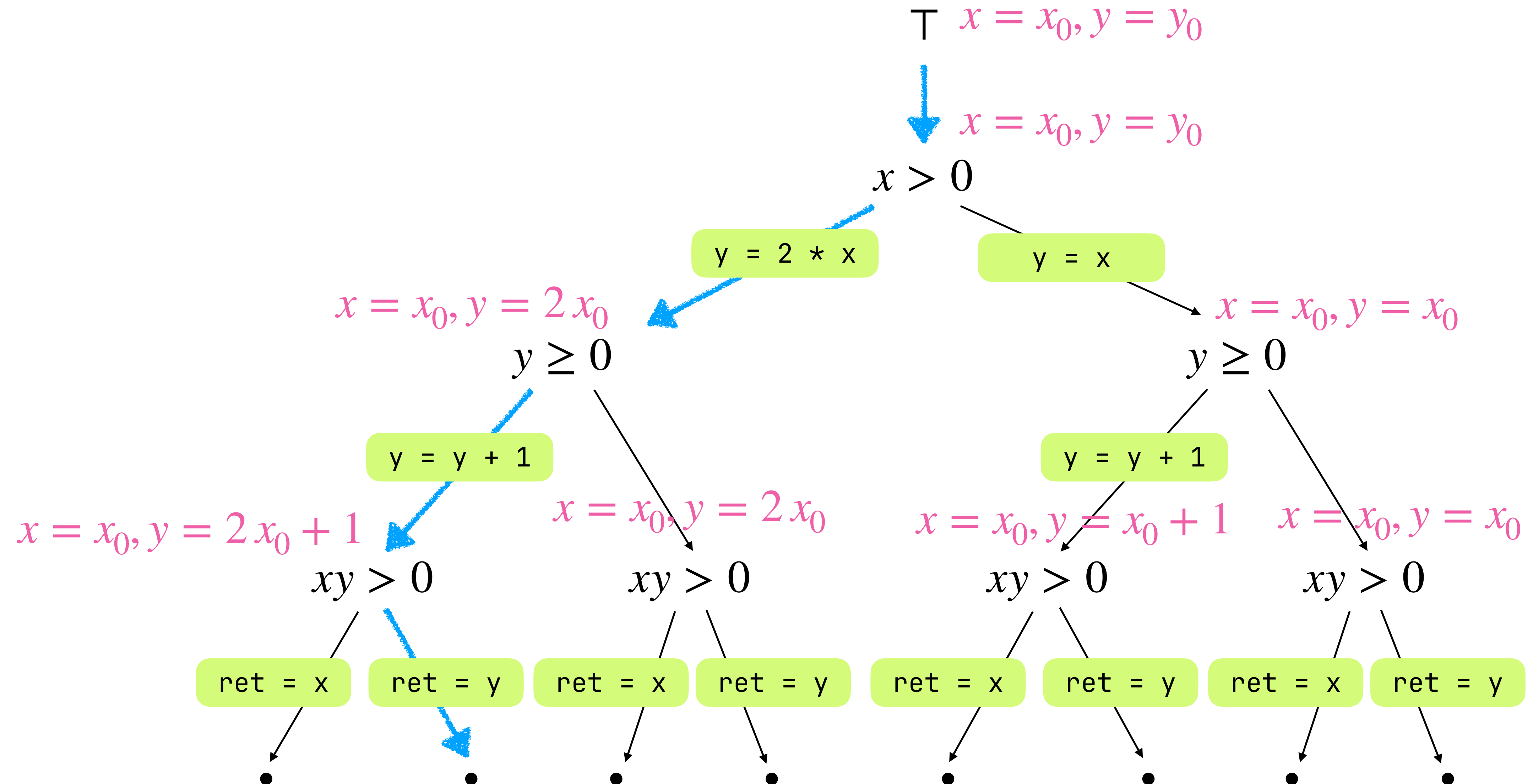
Path Condition:

$$x_0 > 0 \wedge 2x_0 \geq 0 \wedge x_0(2x_0 + 1) > 0$$

Satisfiable



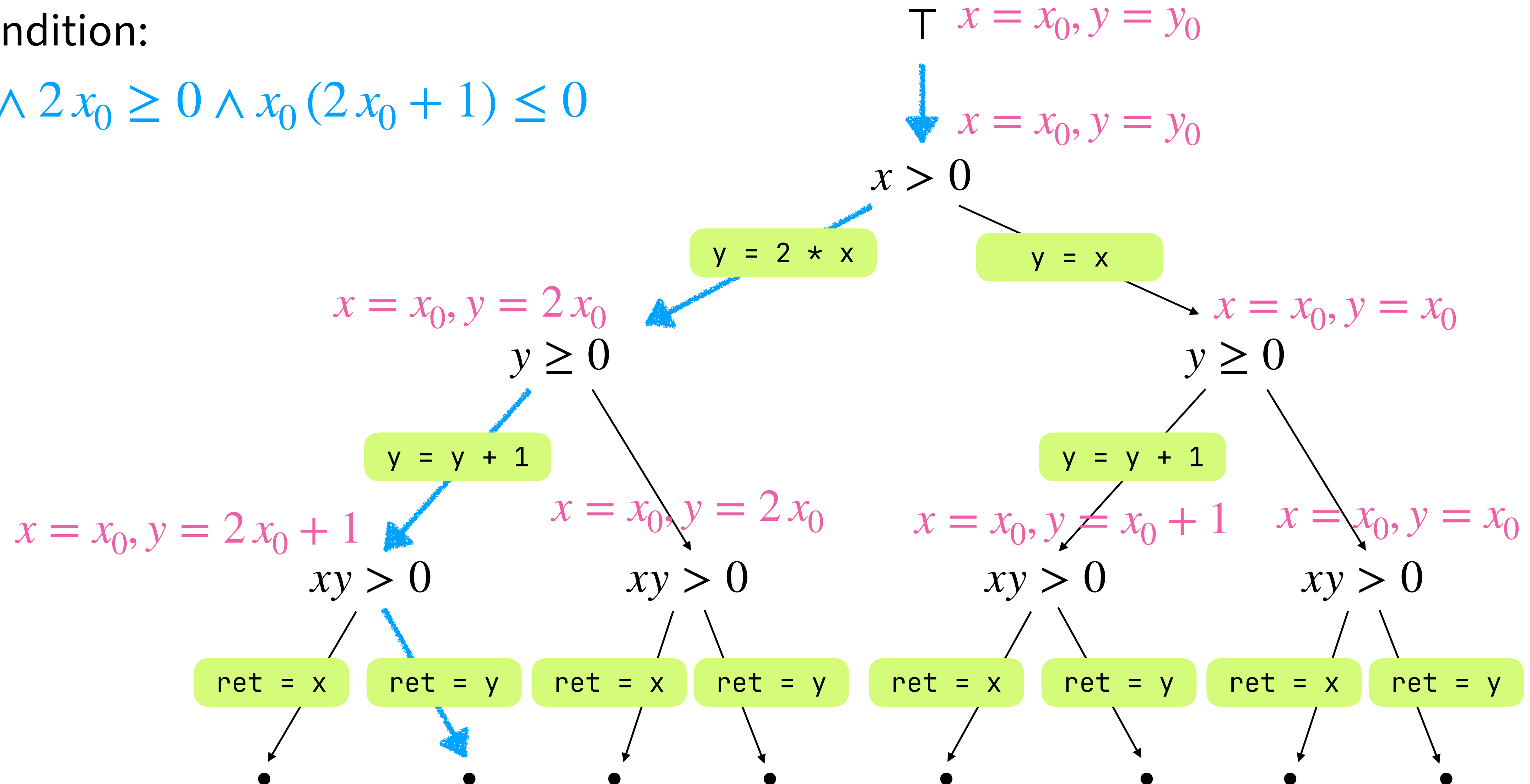
Symbolic Execution: Example



Symbolic Execution: Example

Path Condition:

$$x_0 > 0 \wedge 2x_0 \geq 0 \wedge x_0(2x_0 + 1) \leq 0$$

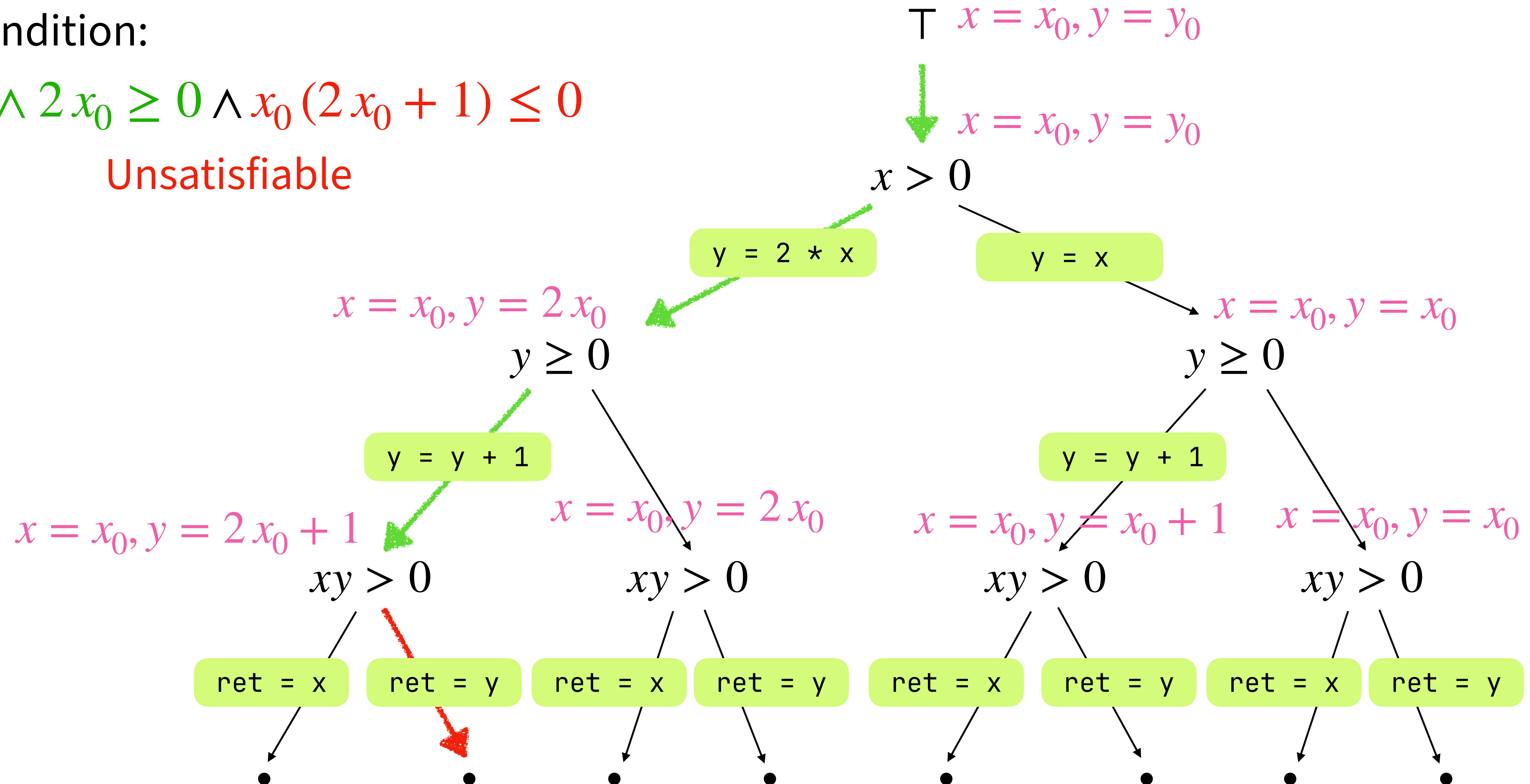


Symbolic Execution: Example

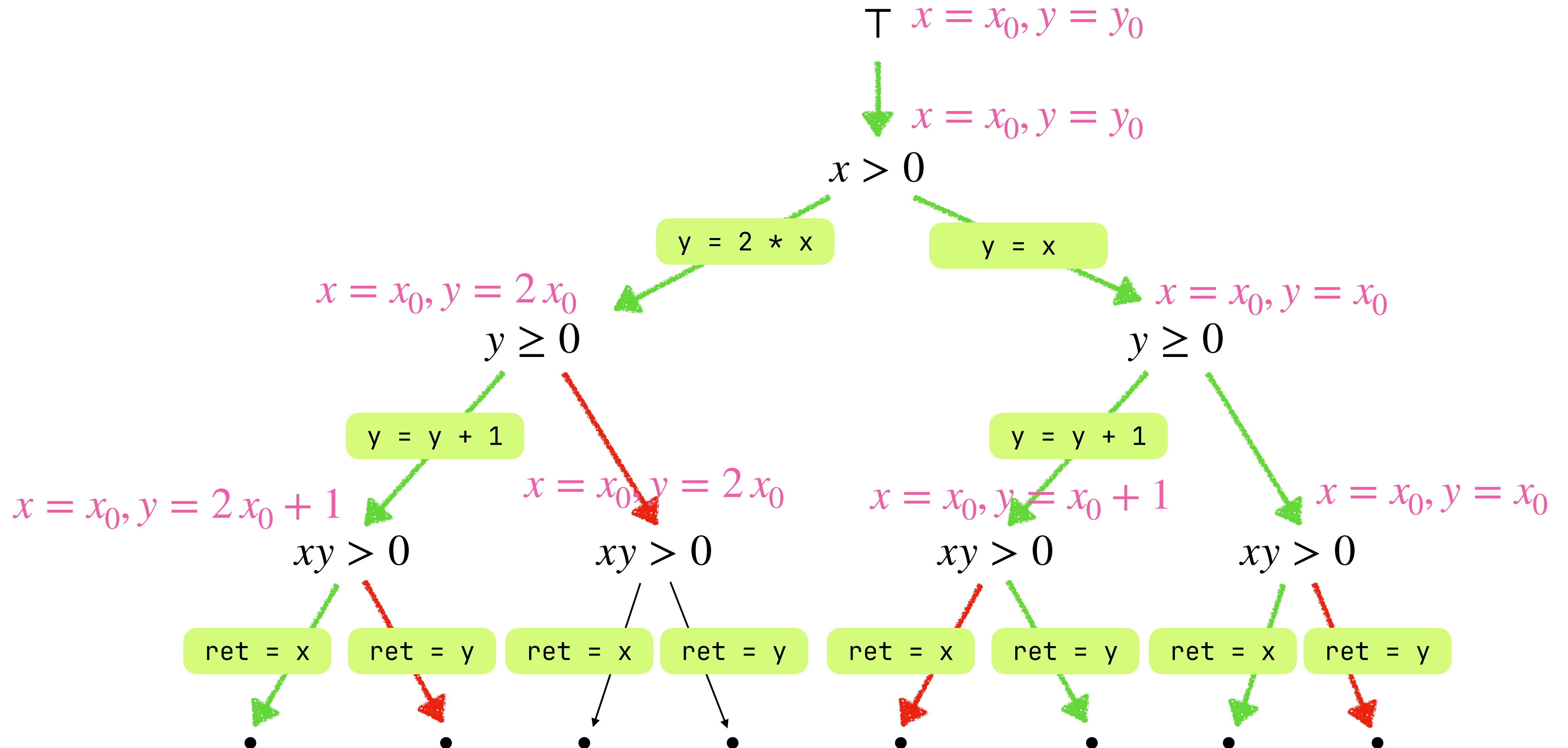
Path Condition:

$$x_0 > 0 \wedge 2x_0 \geq 0 \wedge x_0(2x_0 + 1) \leq 0$$

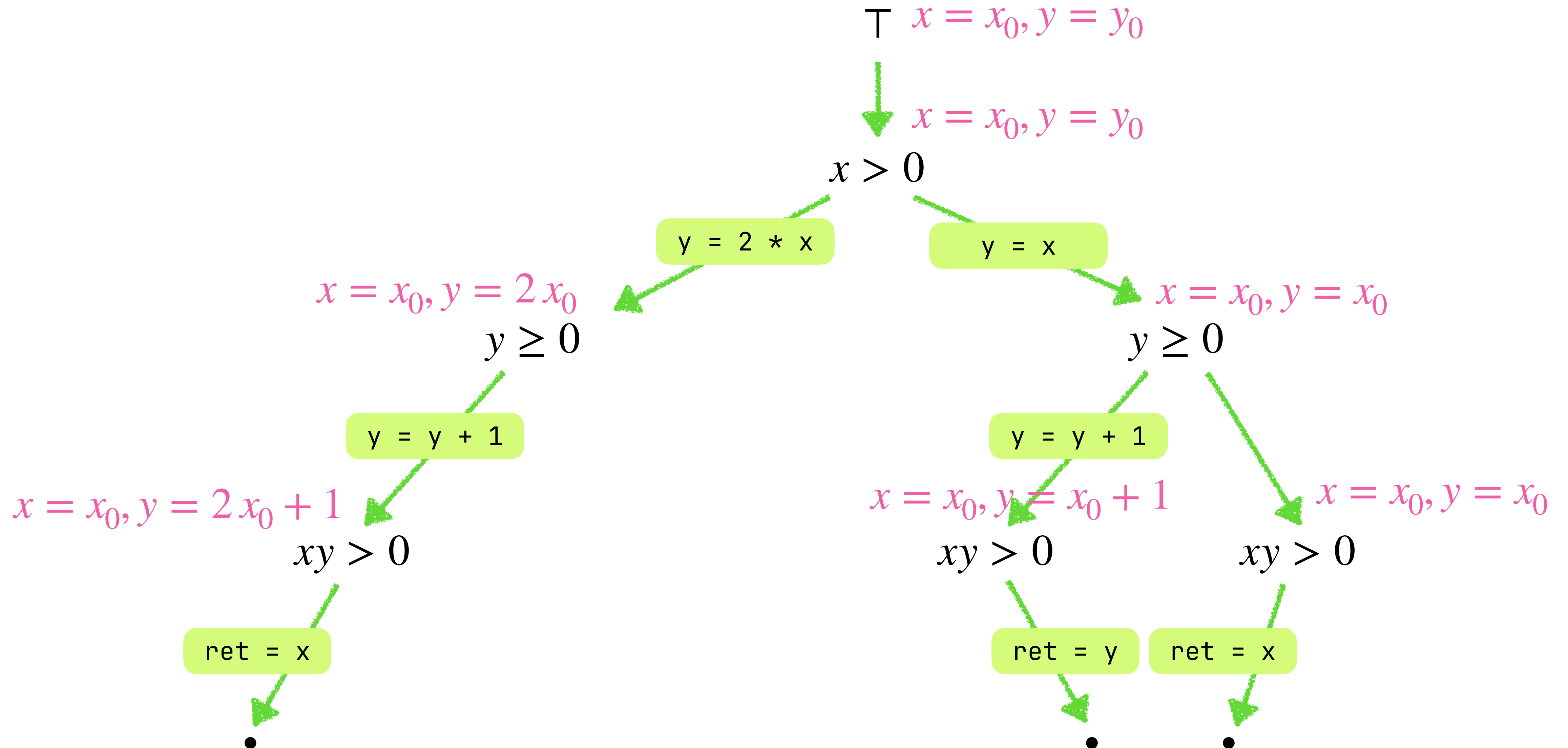
Unsatisfiable



Symbolic Execution: Example

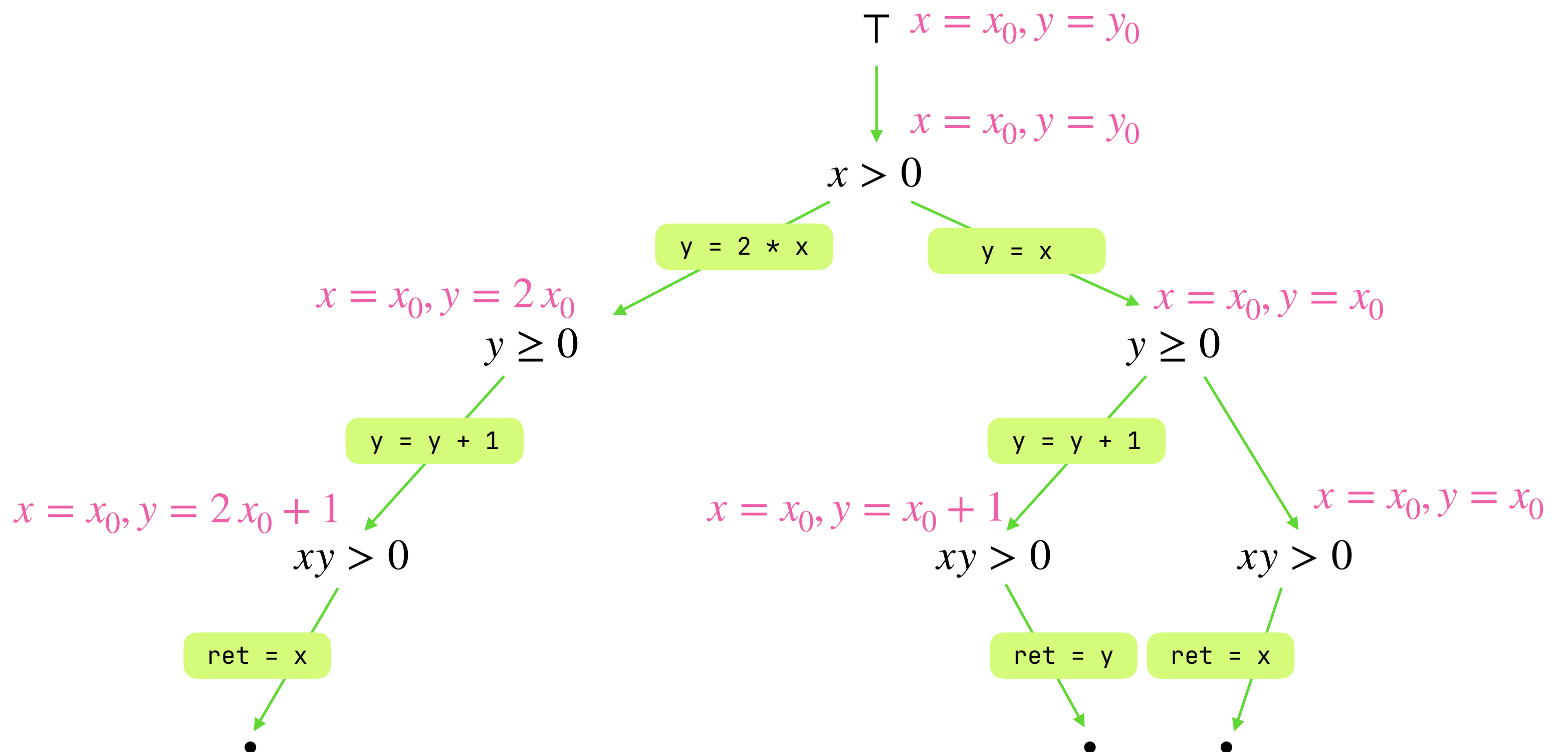


Symbolic Execution: Example



How to use symbolic execution for verification?

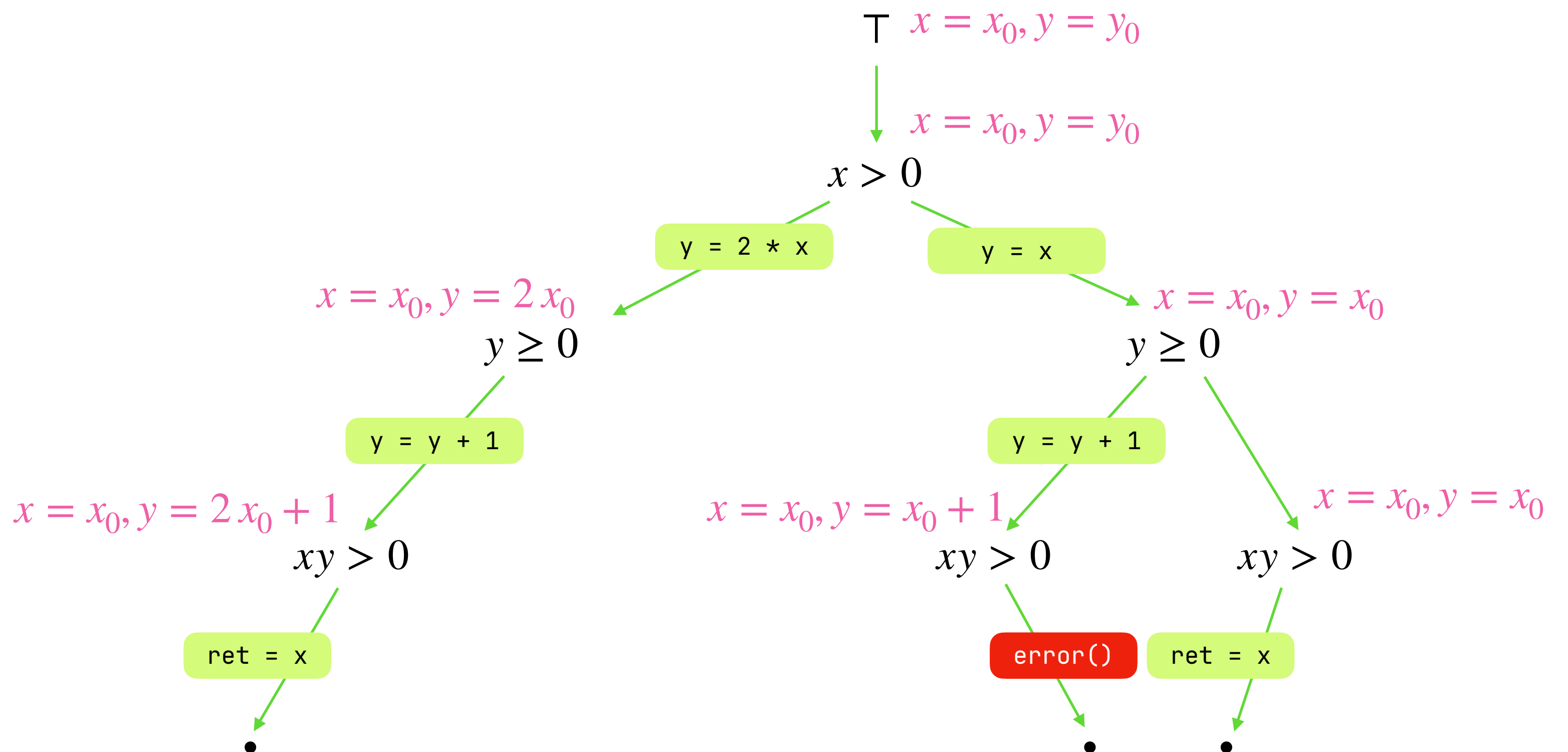
```
def foo(x, y):  
    if x > 0:  
        y = 2 * x  
    else:  
        y = x  
    if y ≥ 0:  
        y = y + 1  
    if x * y > 0:  
        return x  
    else  
        return y
```



How to use symbolic execution for verification?

```
def foo(x, y):  
    if x > 0:  
        y = 2 * x  
    else:  
        y = x  
    if y ≥ 0:  
        y = y + 1  
    if x * y > 0:  
        return x  
    else
```

error()



Error is reachable!

Symbolic Execution v.s. Concrete Executing (a.k.a. Testing)

Symbolic Execution v.s. Concrete Executing (a.k.a. Testing)

Symbolic execution

- Ideal case
 - Every path is enumerated exactly once
 - Guaranteed to find bugs
- In general
 - (+) No path is explored multiple times
 - (+) Can prove the absence of bugs
 - (-) Can be hard to scale
 - (-) Can raise false alarms

Concrete execution

- (-) Paths can be explored zero or multiple times
- (-) Cannot prove the absence of bugs
- (+) Scalable
- (+) Does not raise false alarms

Concolic Testing

An attempt to achieve the best of the both worlds

Concrete + Symbolic

Symbolic execution

- Ideal case
 - Every path is enumerated exactly once
 - Guaranteed to find bugs
- In general
 - (+) No path is explored multiple times
 - (+) Can prove the absence of bugs
 - (-) Can be hard to scale
 - (-) Can raise false alarms

Concrete execution

- (-) Paths can be explored zero or multiple times
- (-) Cannot prove the absence of bugs
- (+) Scalable
- (+) Does not raise false alarms

Concolic Testing: Algorithm

Concolic Testing: Algorithm

1. Sample random concrete input

Concolic Testing: Algorithm

1. Sample random concrete input
2. Run the program with both concrete and symbolic values

Concolic Testing: Algorithm

1. Sample random concrete input
2. Run the program with both concrete and symbolic values
3. Find the deepest branch point where the other branch is **unexplored** and corresponds to a **feasible** path condition

Concolic Testing: Algorithm

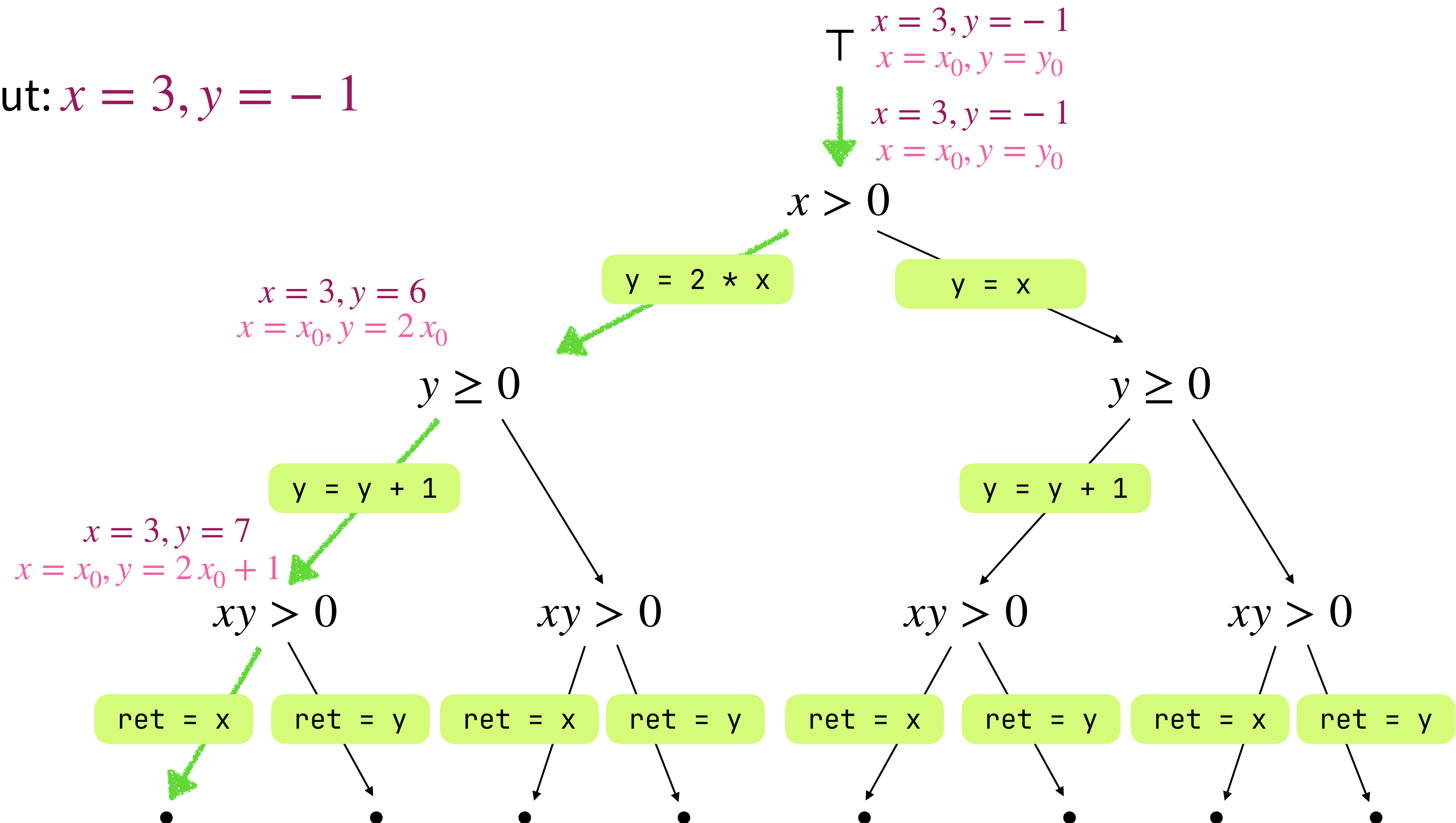
1. Sample random concrete input
2. Run the program with both concrete and symbolic values
3. Find the deepest branch point where the other branch is **unexplored** and corresponds to a **feasible** path condition
 - Return if no such point

Concolic Testing: Algorithm

1. Sample random concrete input
2. Run the program with both concrete and symbolic values
3. Find the deepest branch point where the other branch is **unexplored** and corresponds to a **feasible** path condition
 - Return if no such point
4. Sample an input that satisfies the path condition, and return to Step 2

Concolic Testing: Example 1

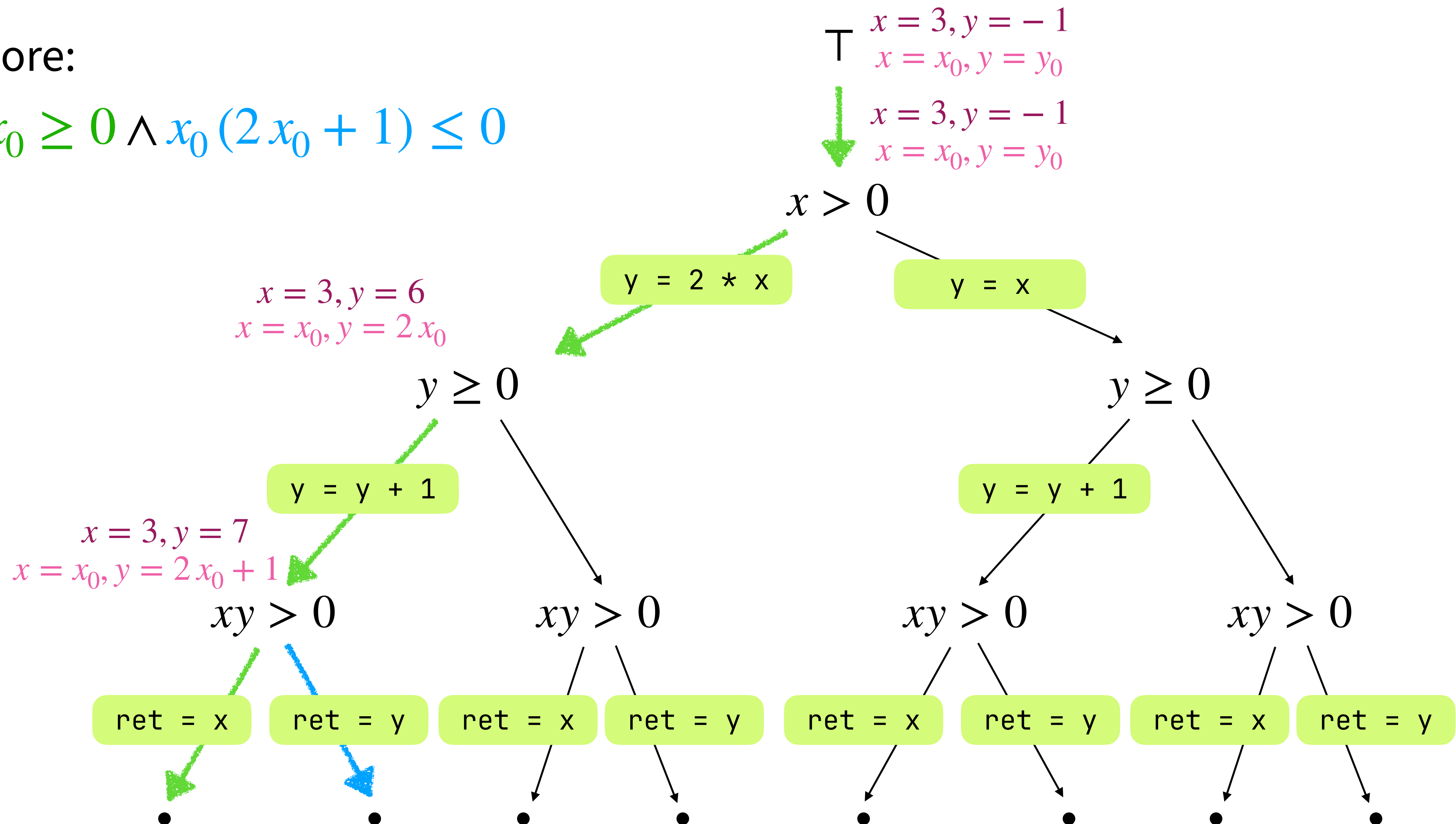
Random input: $x = 3, y = -1$



Concolic Testing: Example 1

Want to explore:

$$x_0 > 0 \wedge 2x_0 \geq 0 \wedge x_0(2x_0 + 1) \leq 0$$

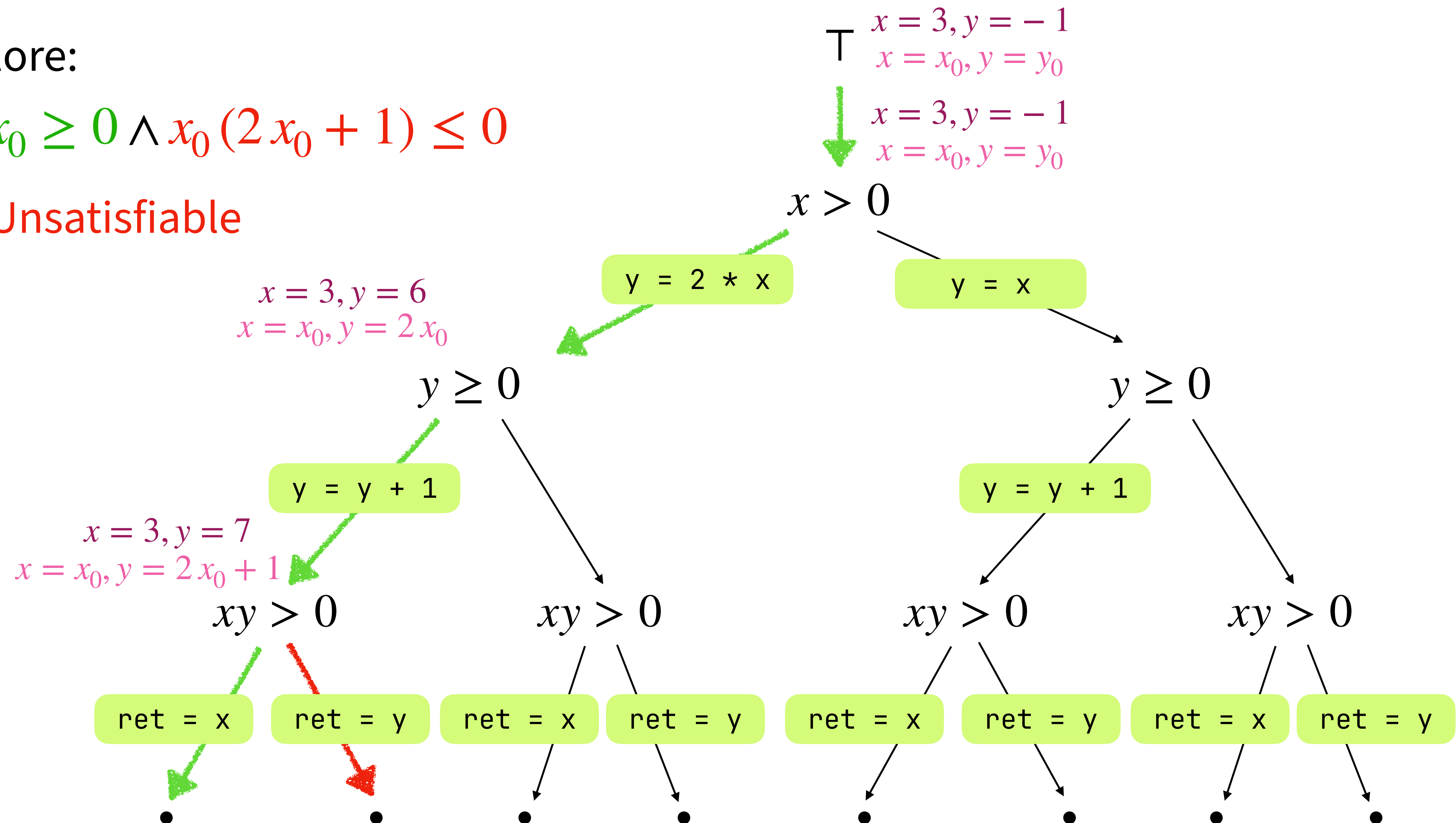


Concolic Testing: Example 1

Want to explore:

$$x_0 > 0 \wedge 2x_0 \geq 0 \wedge x_0(2x_0 + 1) \leq 0$$

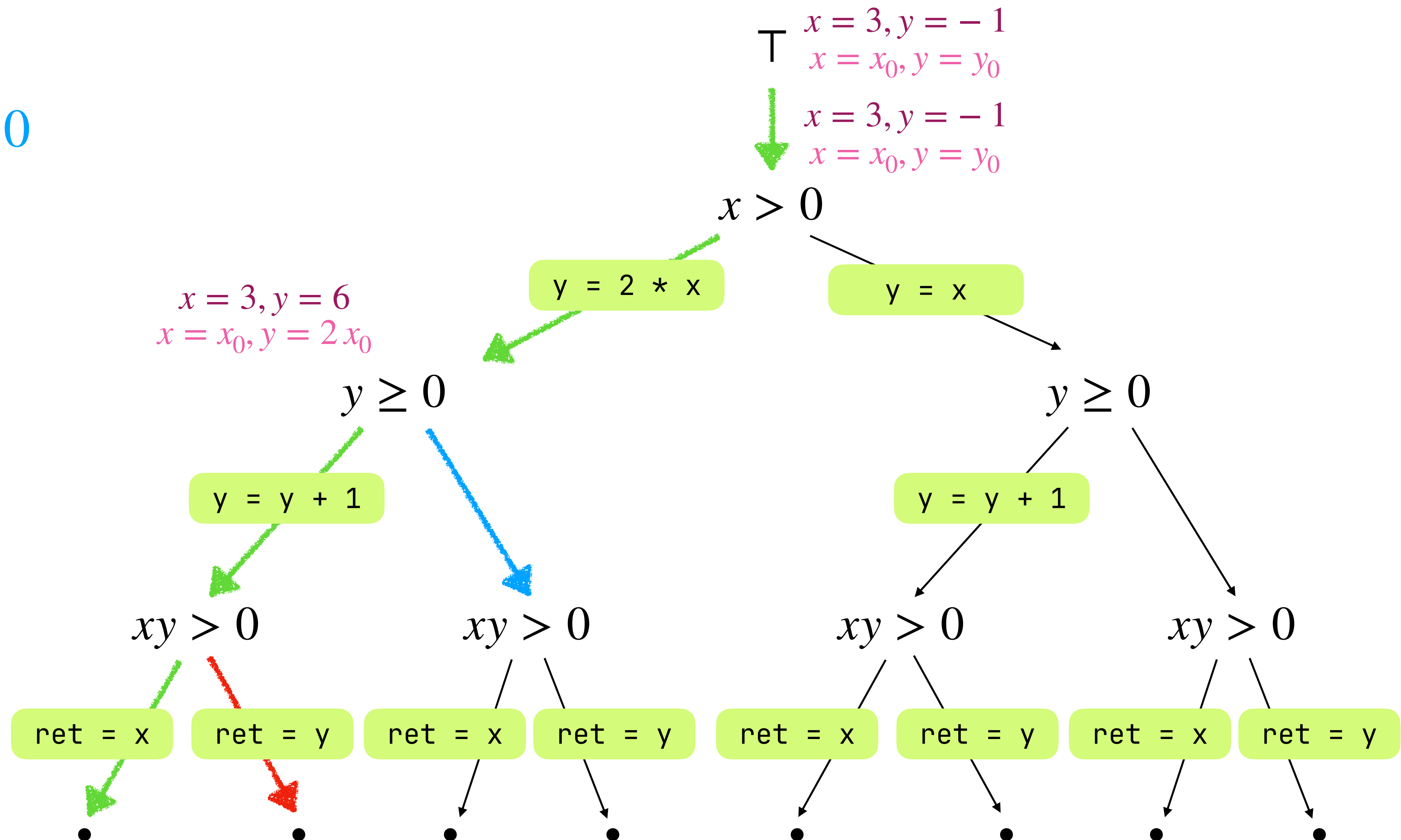
Unsatisfiable



Concolic Testing: Example 1

Want to explore:

$$x_0 > 0 \wedge 2x_0 < 0$$

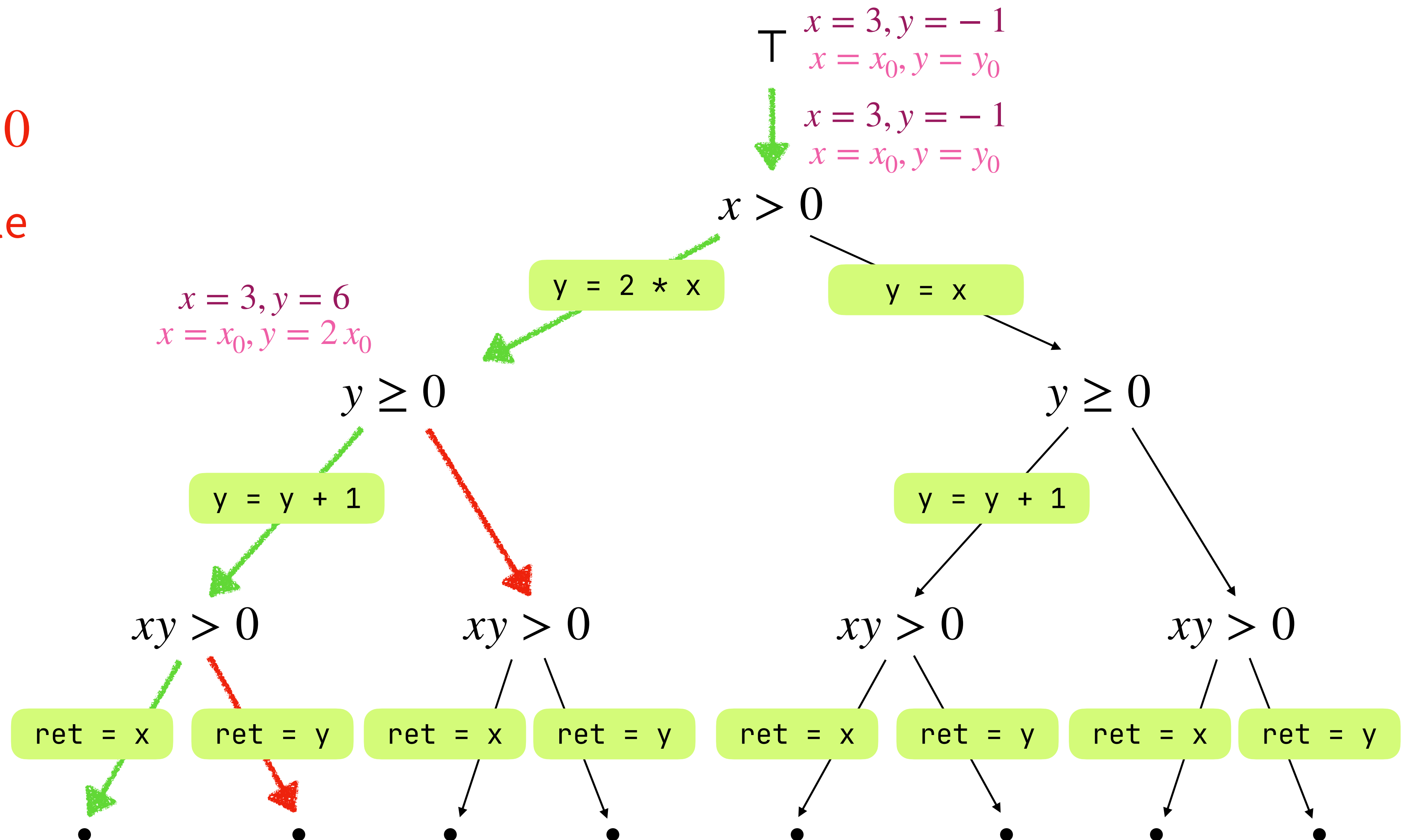


Concolic Testing: Example 1

Want to explore:

$$x_0 > 0 \wedge 2x_0 < 0$$

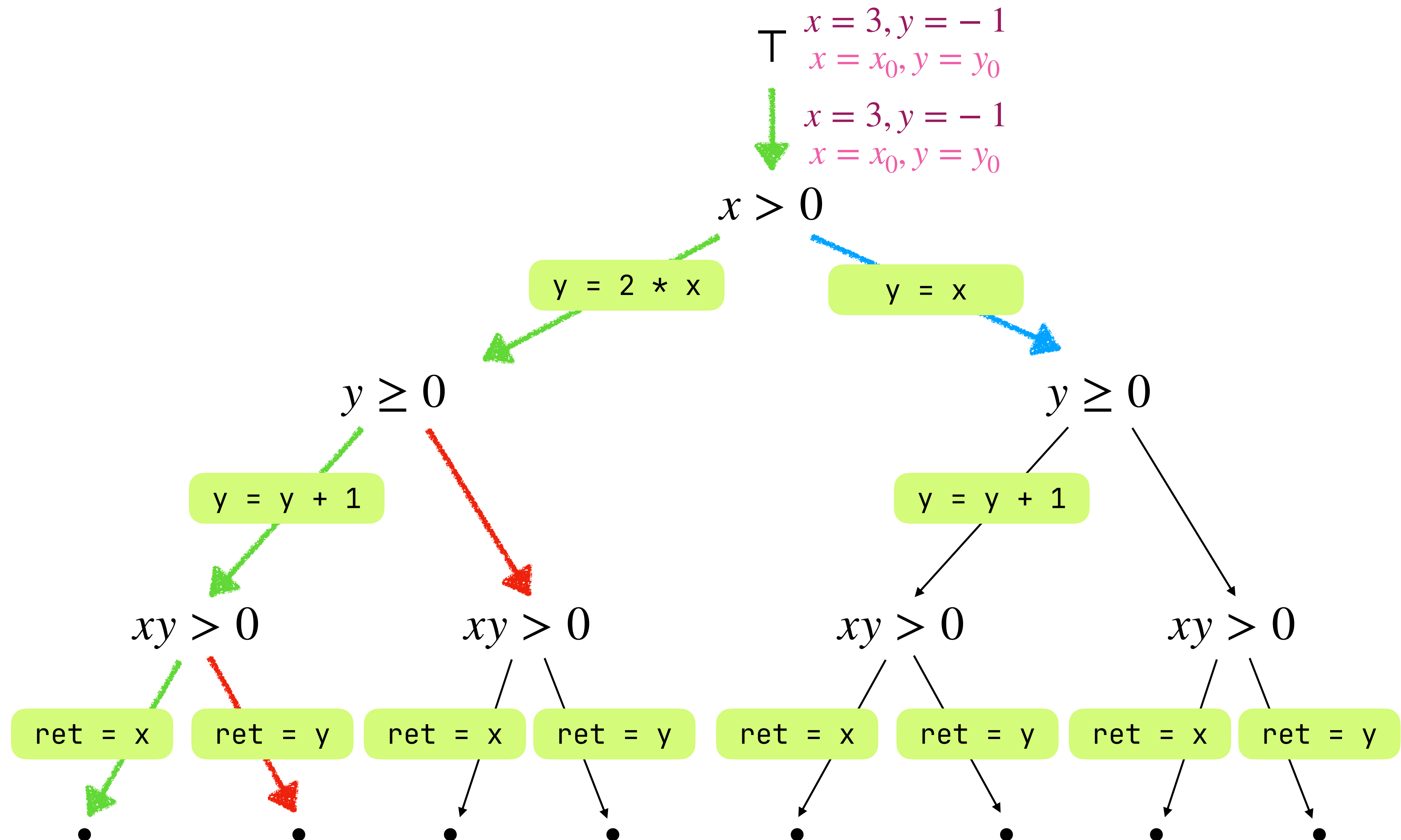
Unsatisfiable



Concolic Testing: Example 1

Want to explore:

$$x_0 \leq 0$$

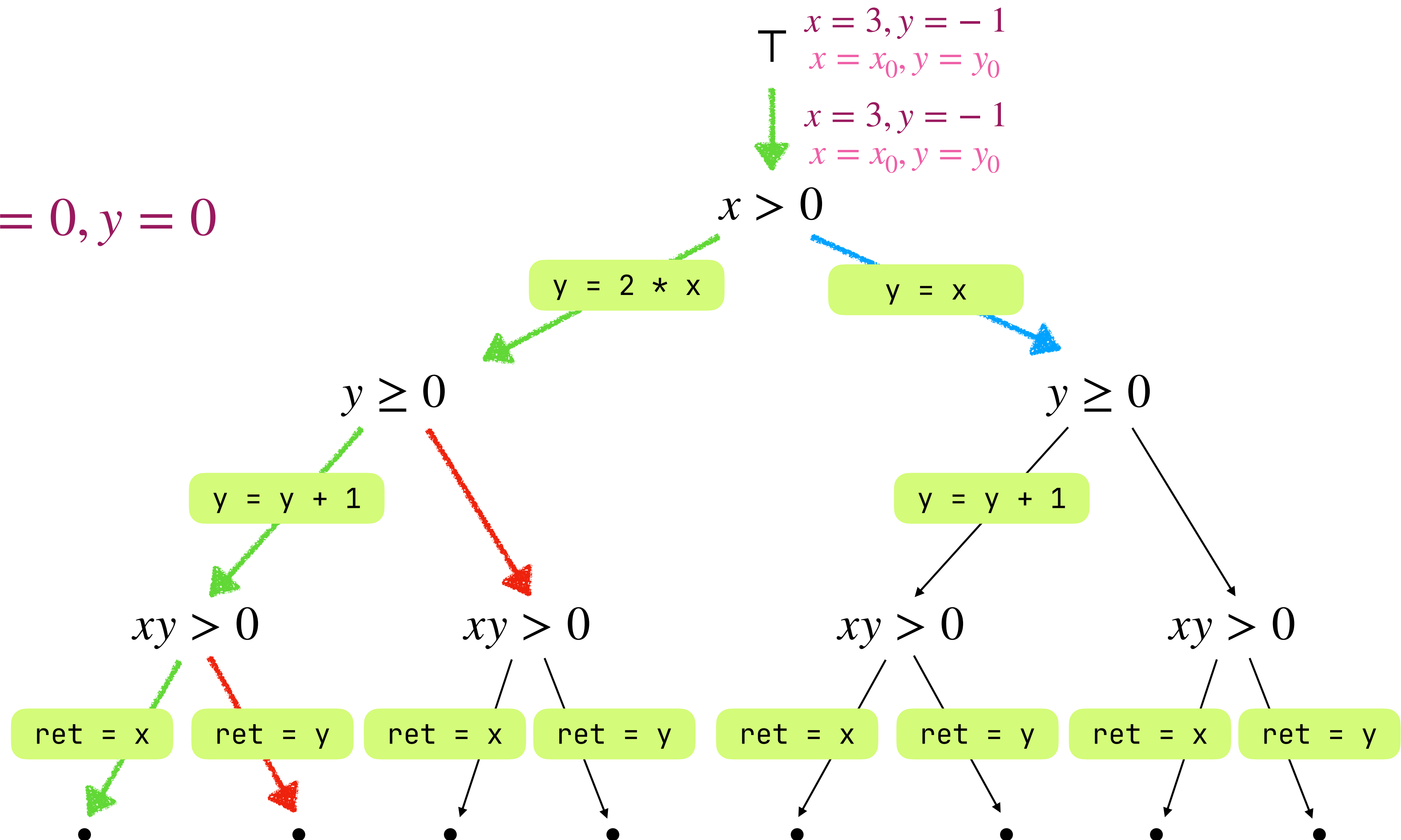


Concolic Testing: Example 1

Want to explore:

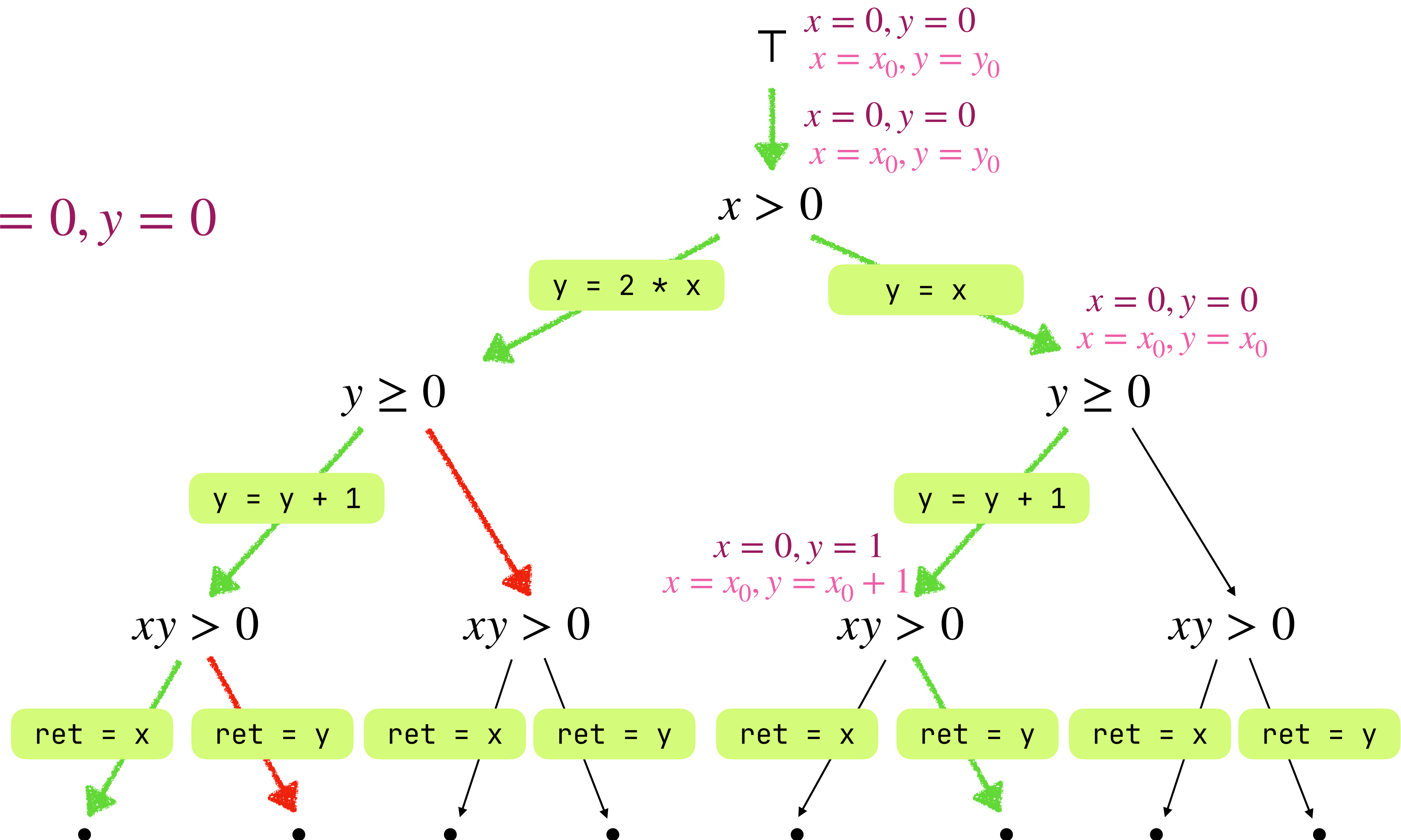
$$x_0 \leq 0$$

Sample input: $x = 0, y = 0$



Concolic Testing: Example 1

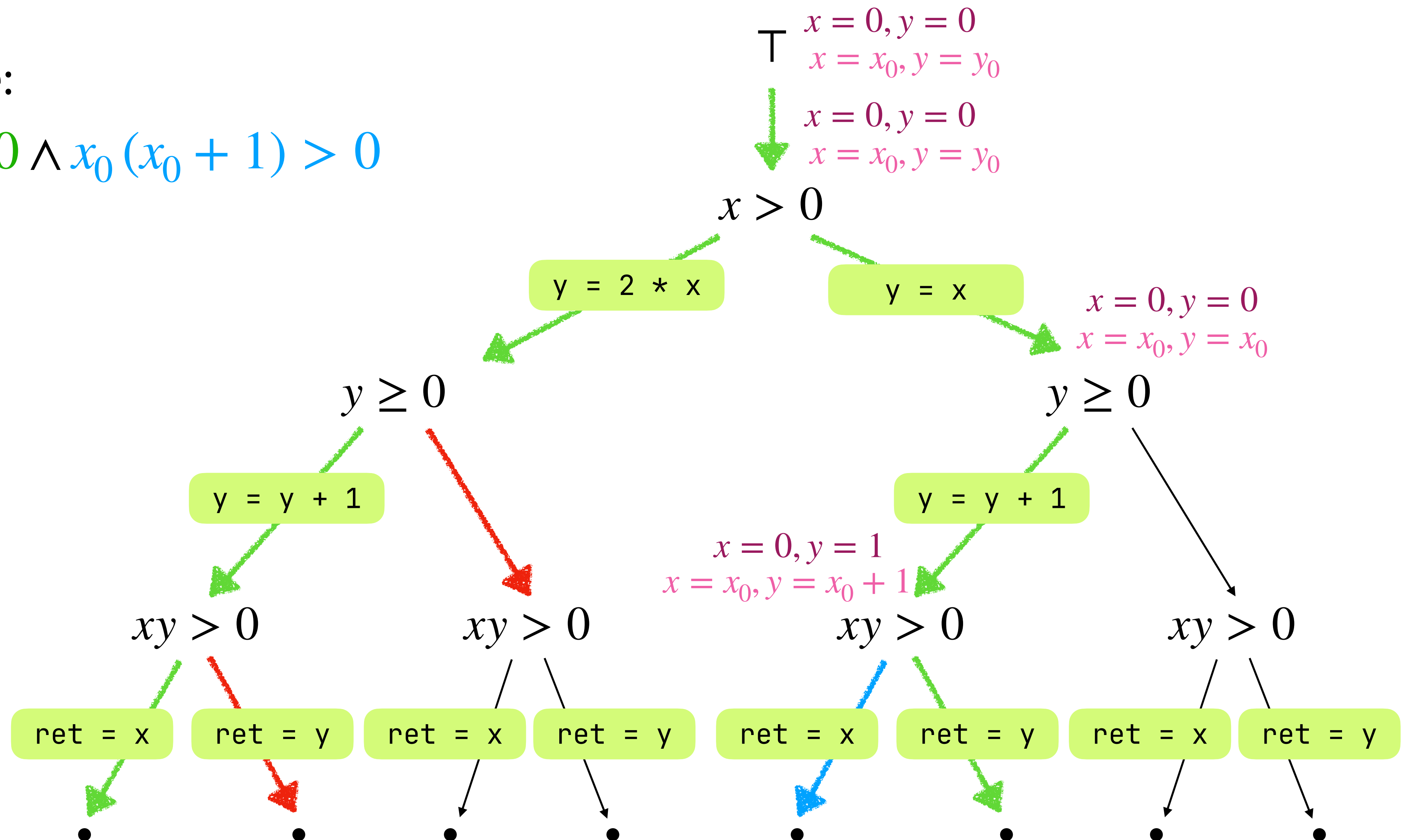
Sample input: $x = 0, y = 0$



Concolic Testing: Example 1

Want to explore:

$$x_0 \leq 0 \wedge x_0 \geq 0 \wedge x_0(x_0 + 1) > 0$$

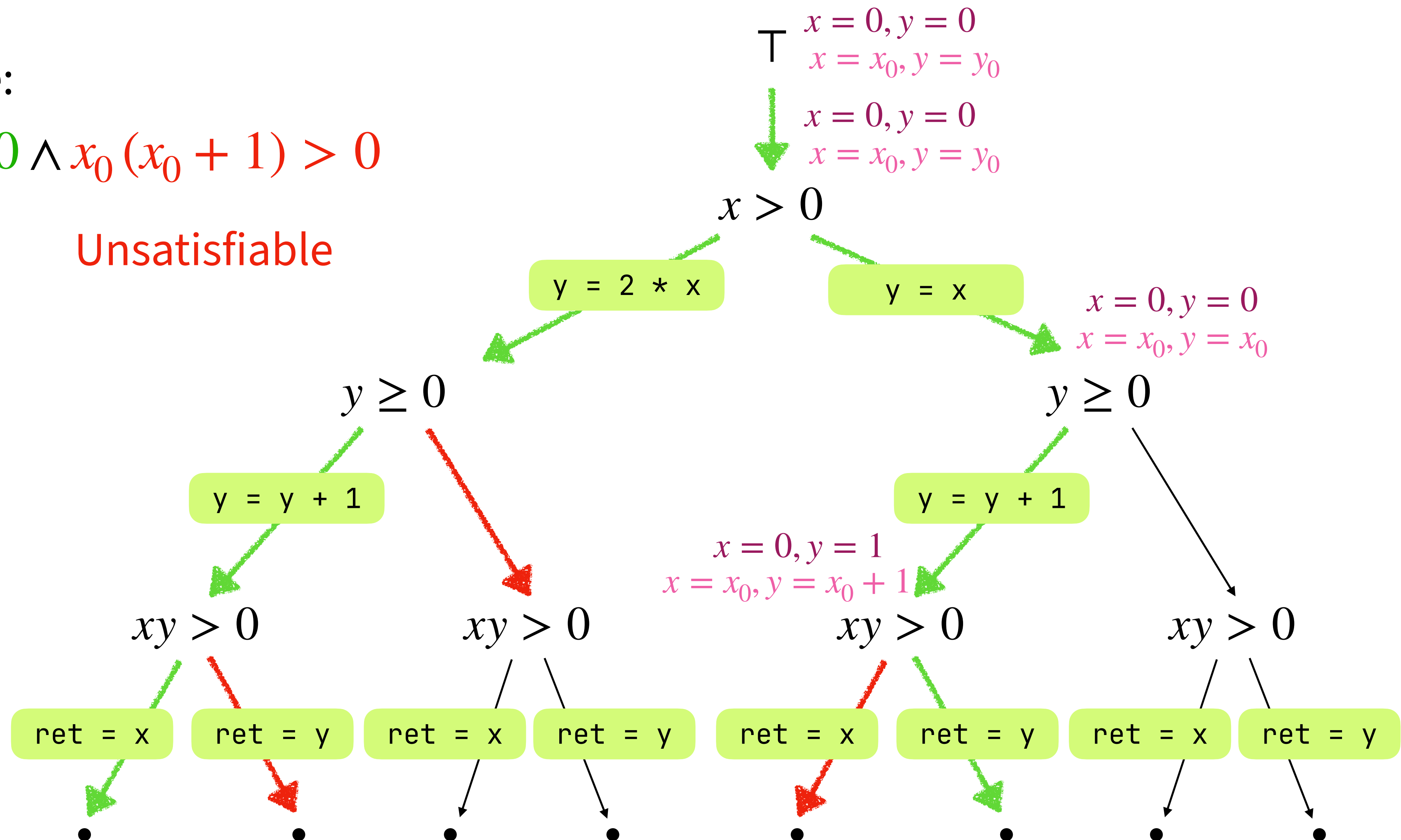


Concolic Testing: Example 1

Want to explore:

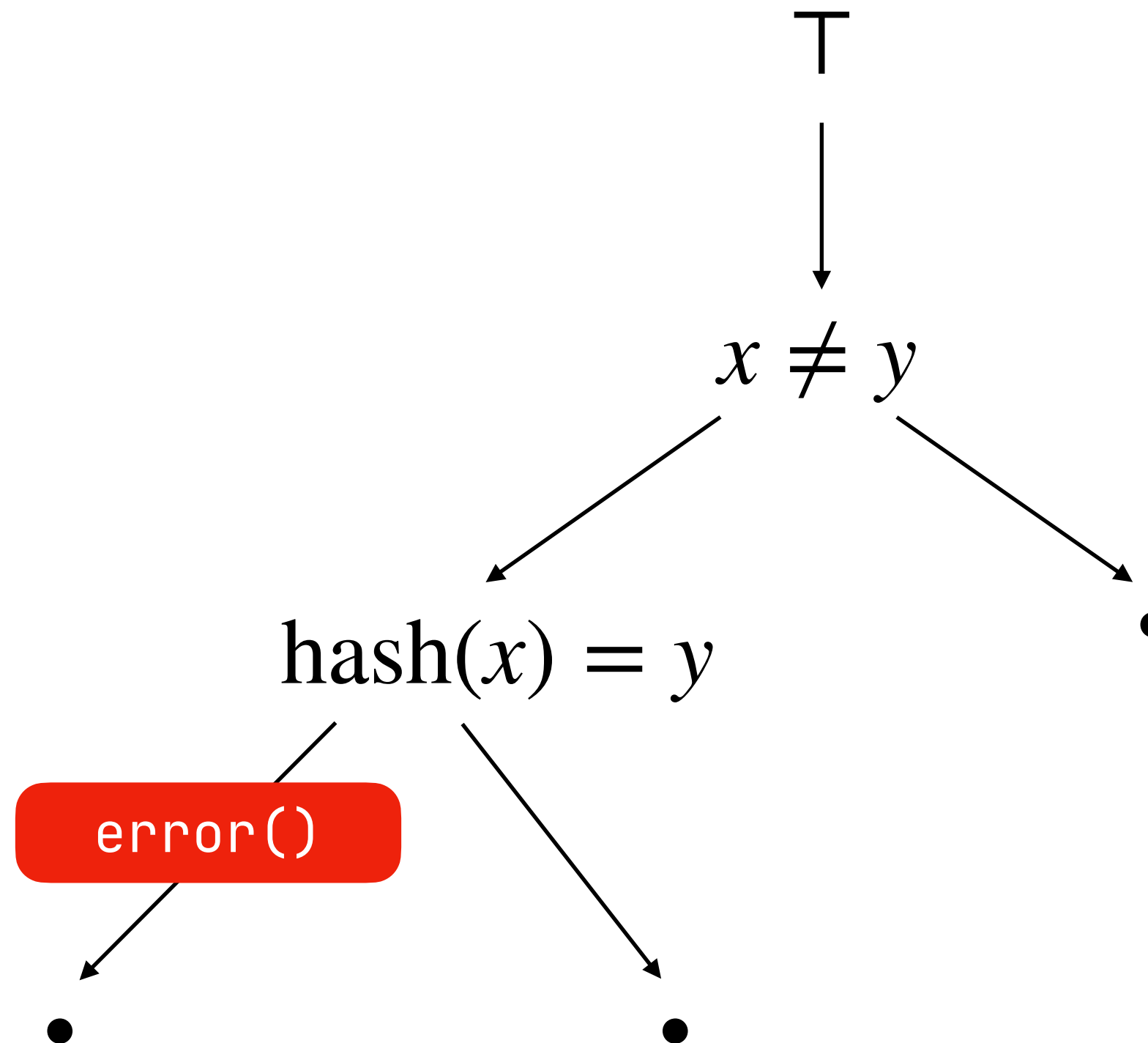
$$x_0 \leq 0 \wedge x_0 \geq 0 \wedge x_0 (x_0 + 1) > 0$$

Unsatisfiable



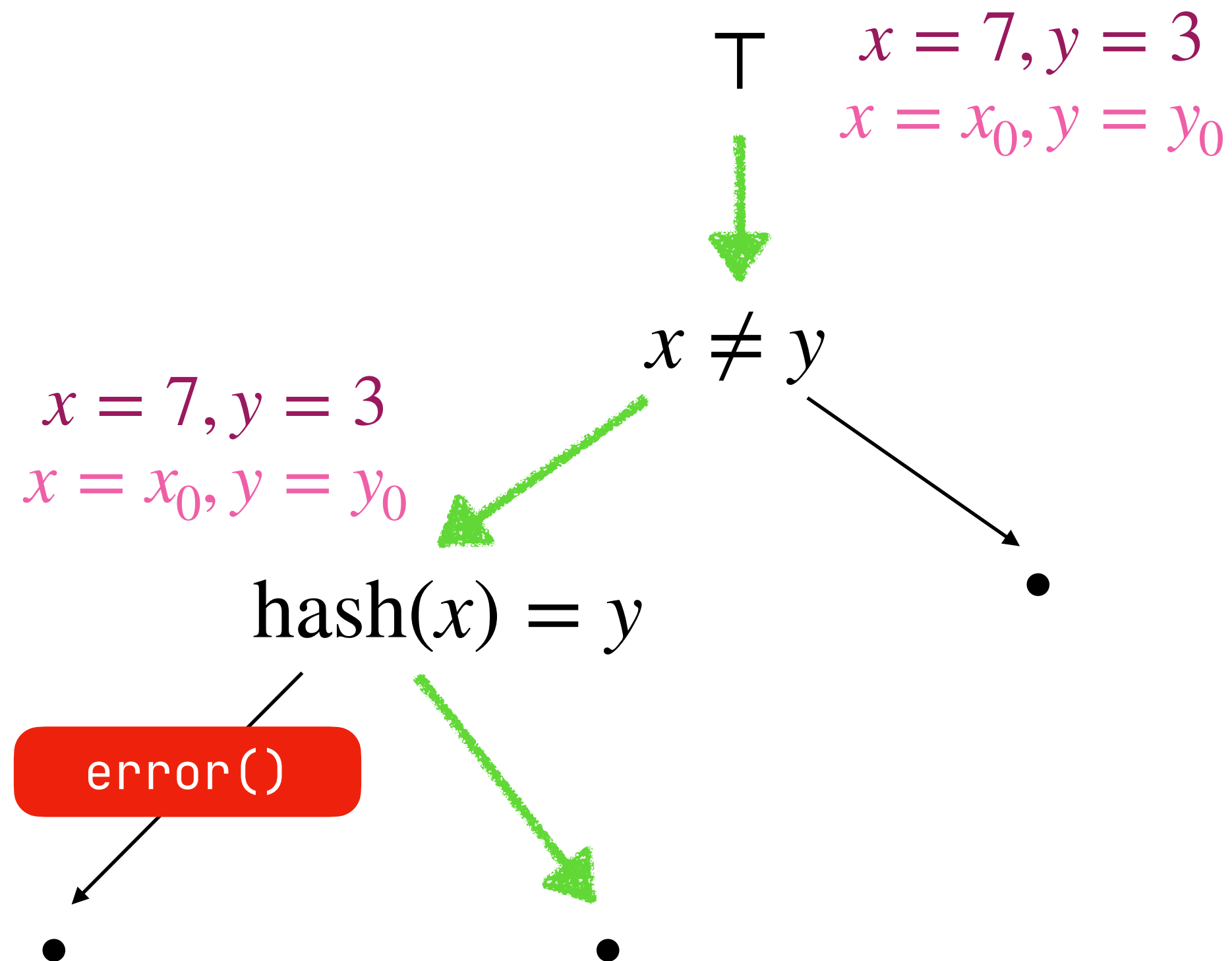
Concolic Testing: Example 2

```
def bar(x, y):  
    if x  $\neq$  y:  
        if hash(x) == y:  
            error()
```



Concolic Testing: Example 2

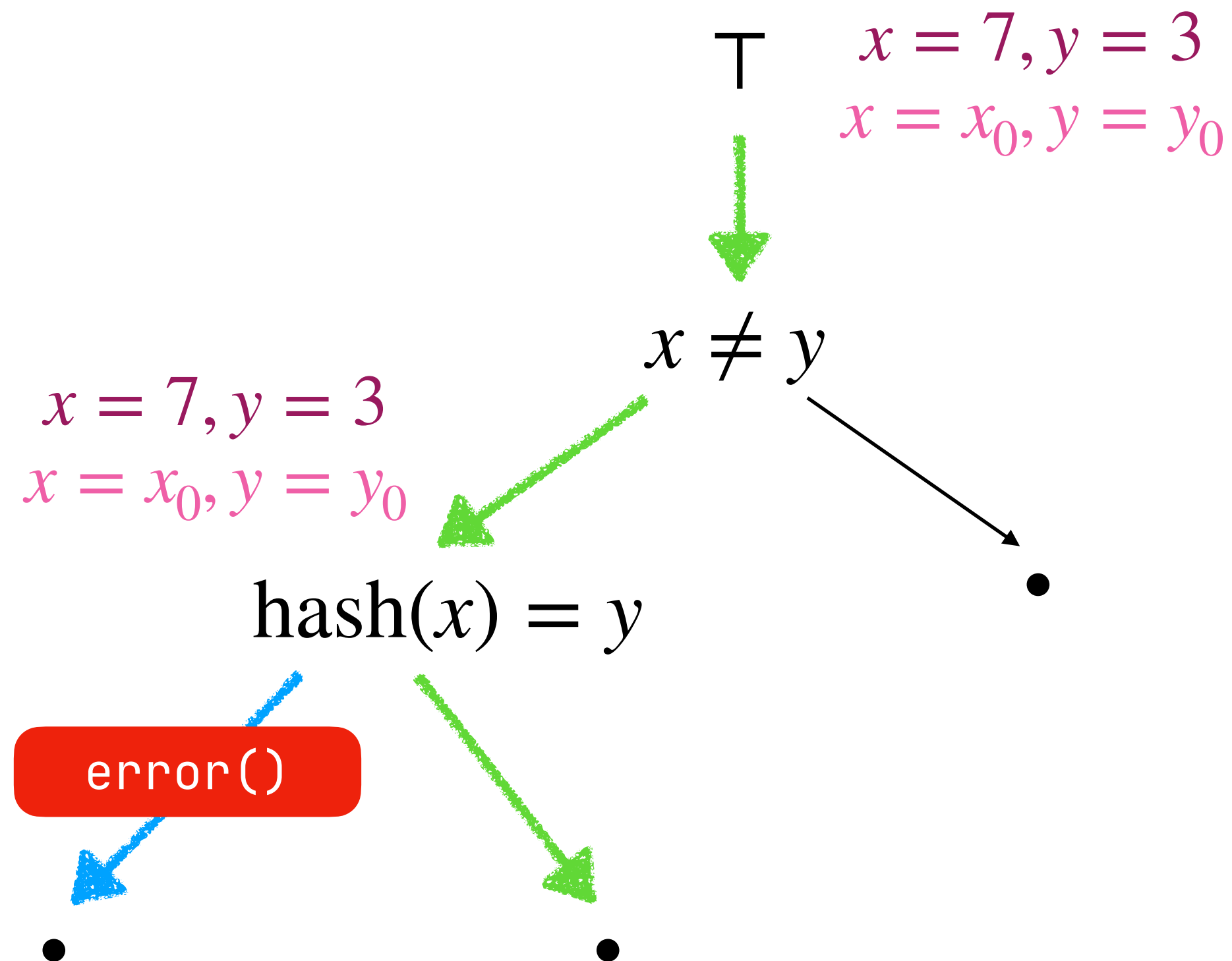
```
def bar(x, y):  
    if x  $\neq$  y:  
        if hash(x) == y:  
            error()
```



Run with random input $x = 7, y = 3$

Concolic Testing: Example 2

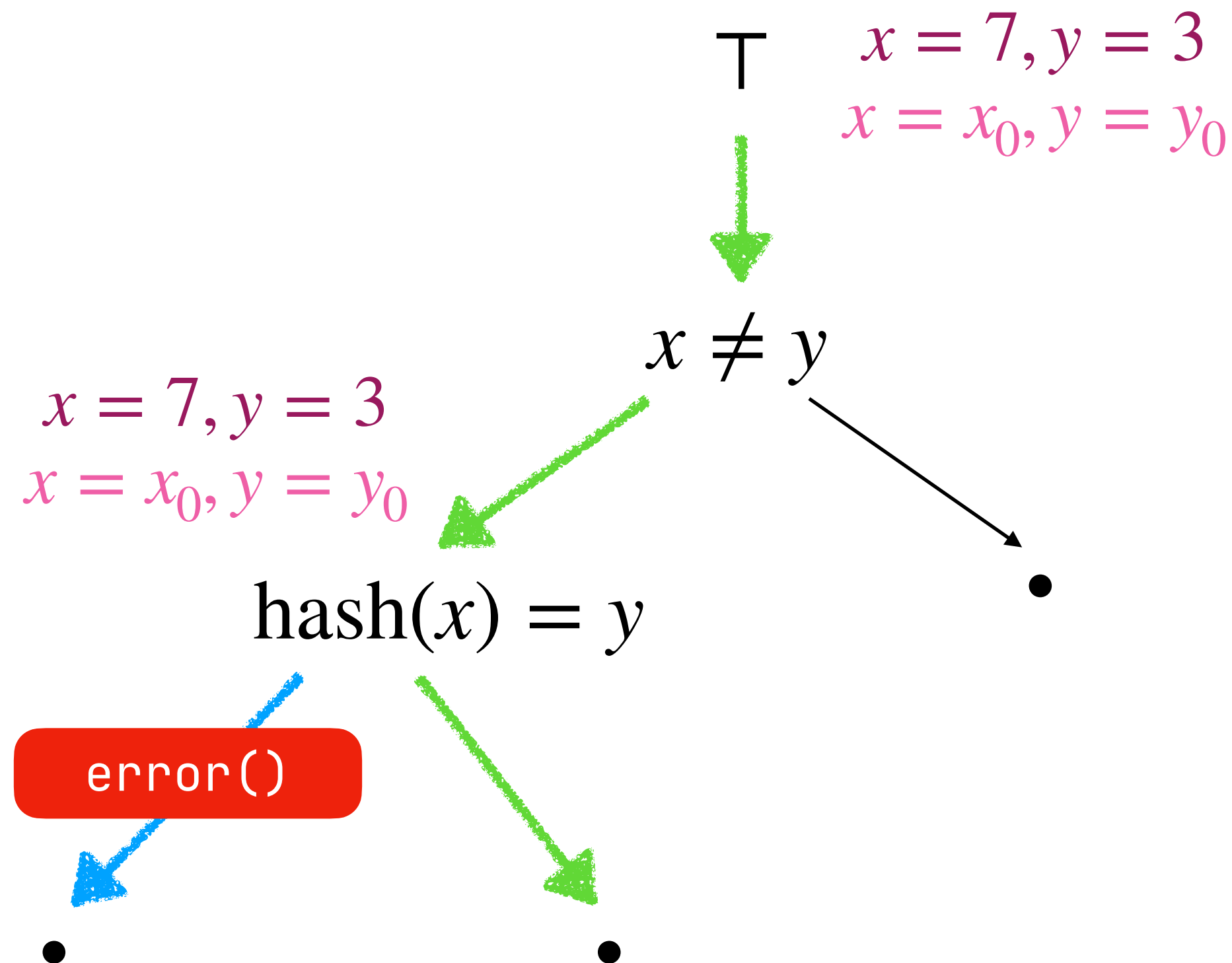
```
def bar(x, y):  
    if x  $\neq$  y:  
        if hash(x) == y:  
            error()
```



Want to explore $x_0 \neq y_0 \wedge \text{hash}(x_0) = y_0$

Concolic Testing: Example 2

```
def bar(x, y):  
    if x  $\neq$  y:  
        if hash(x) == y:  
            error()
```

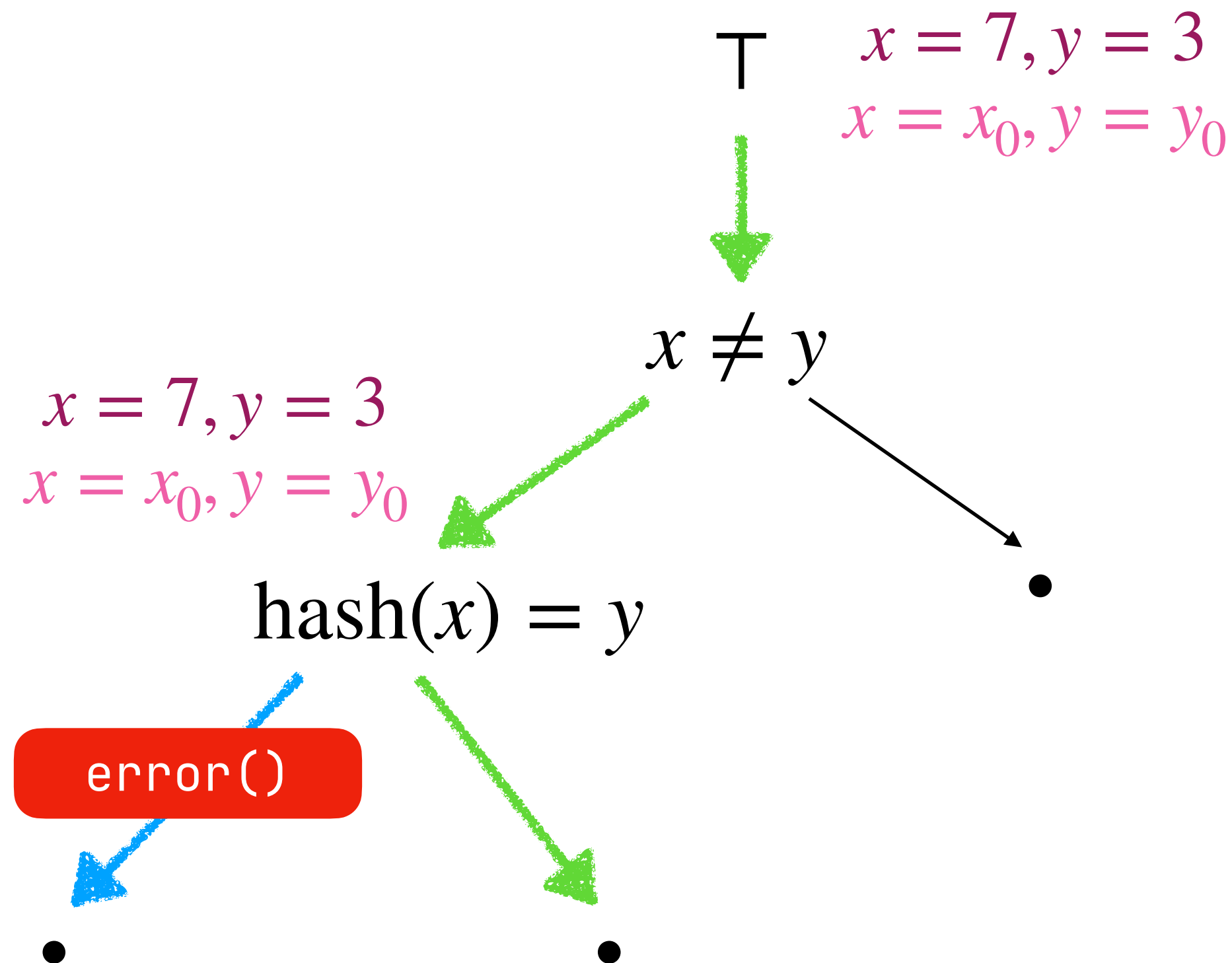


Want to explore $x_0 \neq y_0 \wedge \text{hash}(x_0) = y_0$

hash may be complicated or a blackbox!

Concolic Testing: Example 2

```
def bar(x, y):  
    if x  $\neq$  y:  
        if hash(x) == y:  
            error()
```

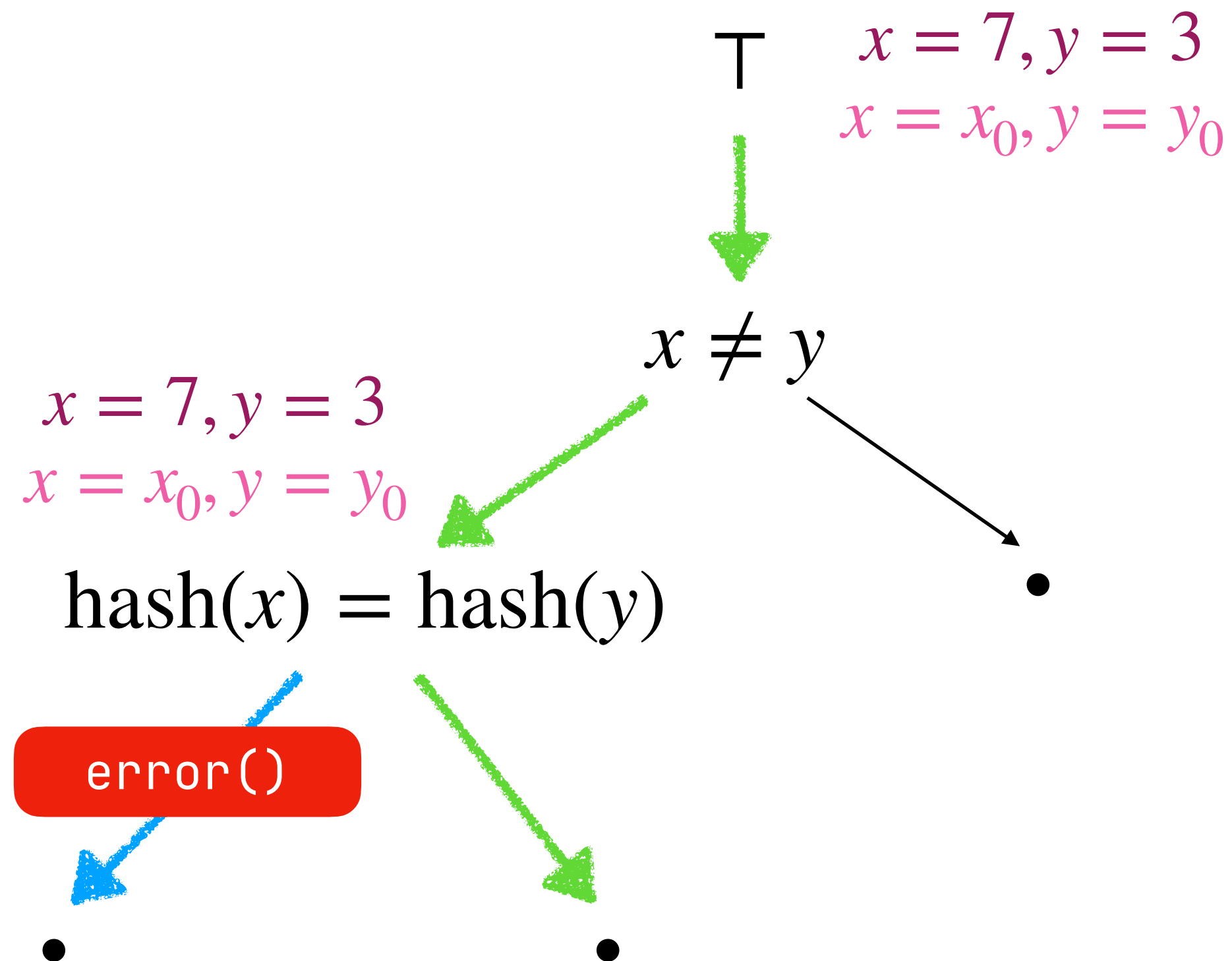


Want to explore $7 \neq y_0 \wedge \text{hash}(7) = y_0$

Use the concrete value of x to simplify the constraints

Concolic Testing: Example 3

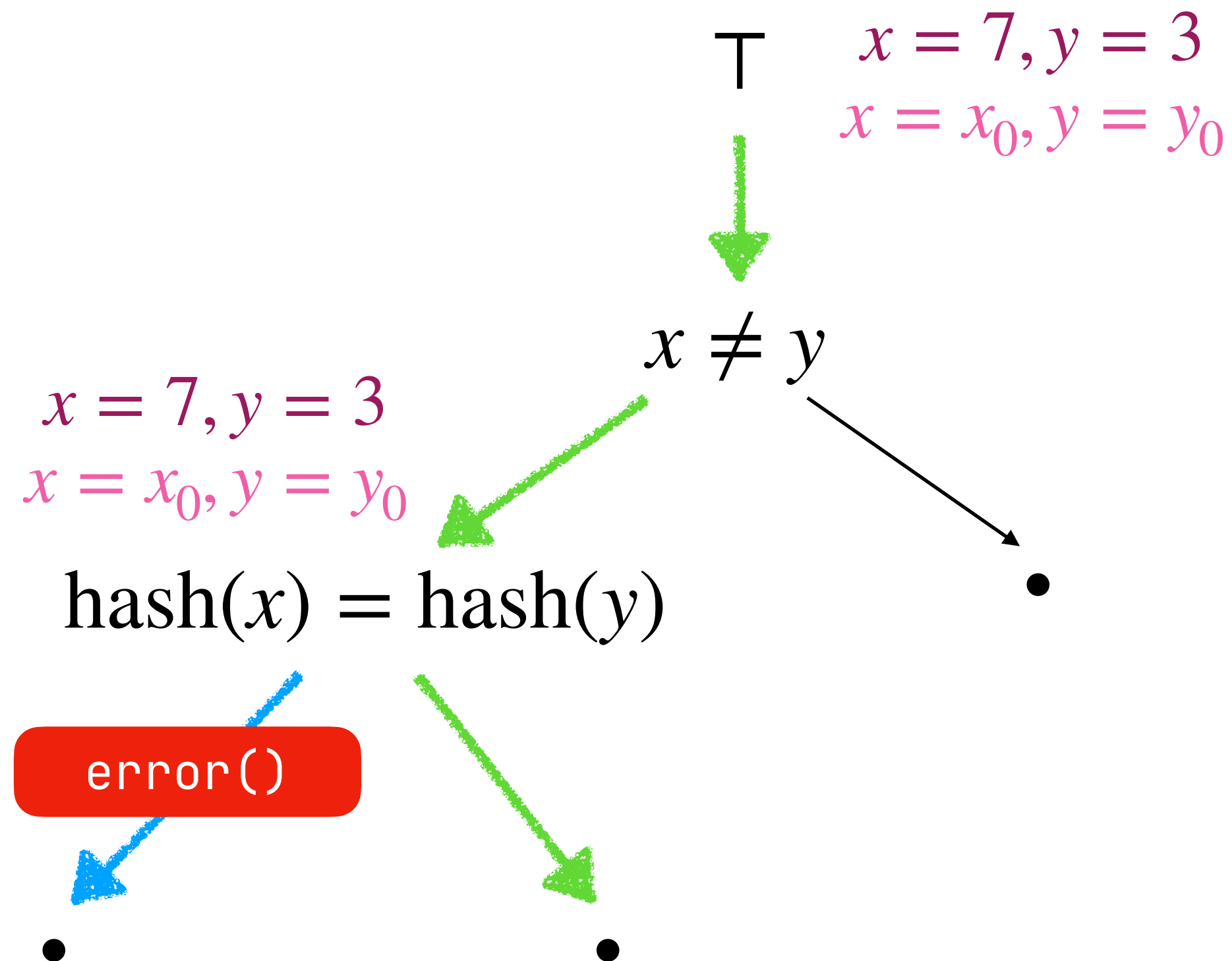
```
def bar(x, y):  
    if x  $\neq$  y:  
        if hash(x) == hash(y):  
            error()
```



Want to explore $x_0 \neq y_0 \wedge \text{hash}(x_0) = \text{hash}(y_0)$

Concolic Testing: Example 3

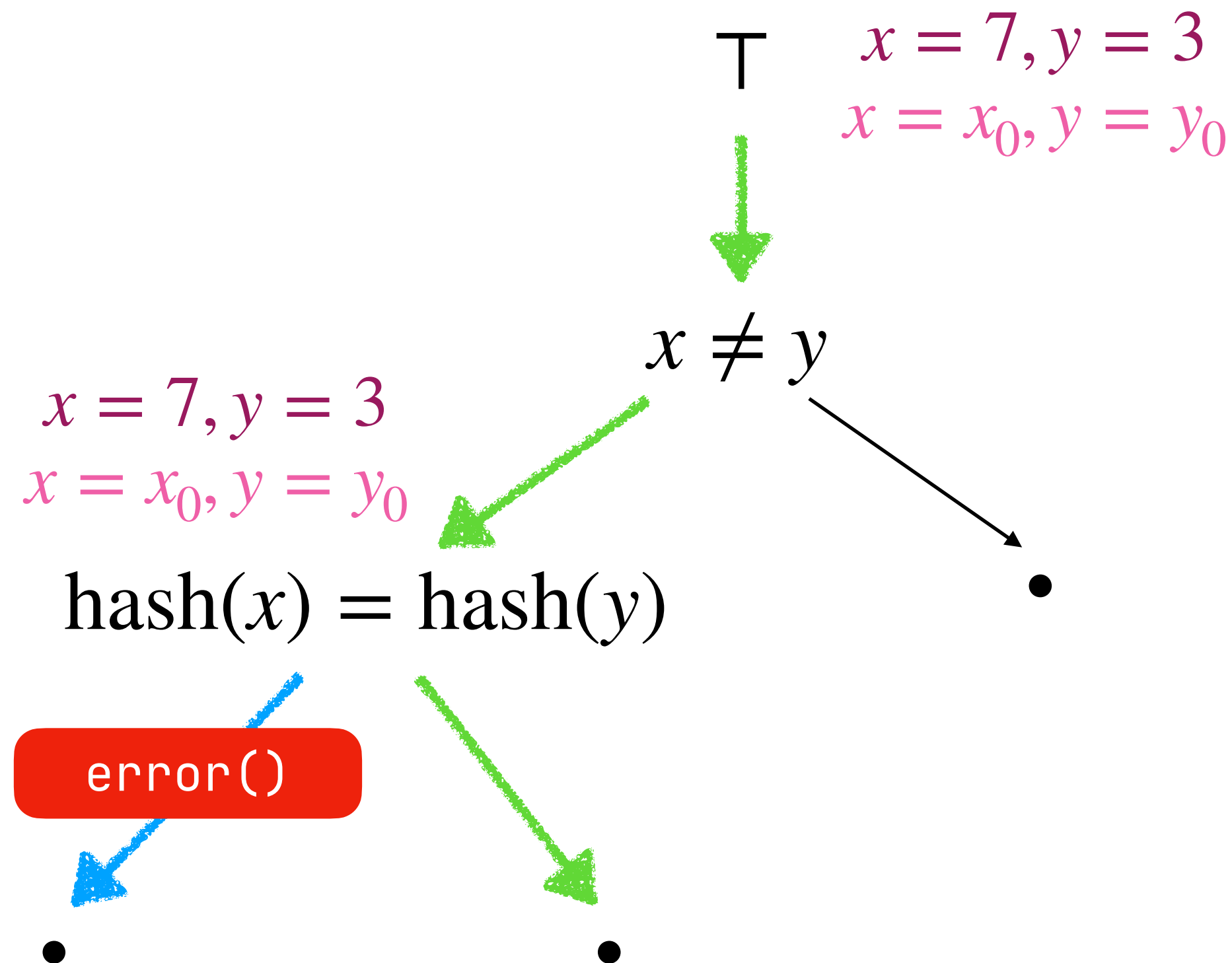
```
def bar(x, y):  
    if x  $\neq$  y:  
        if hash(x) == hash(y):  
            error()
```



Want to explore $7 \neq y_0 \wedge \text{hash}(7) = \text{hash}(y_0)$

Concolic Testing: Example 3

```
def bar(x, y):  
    if x  $\neq$  y:  
        if hash(x) == hash(y):  
            error()
```

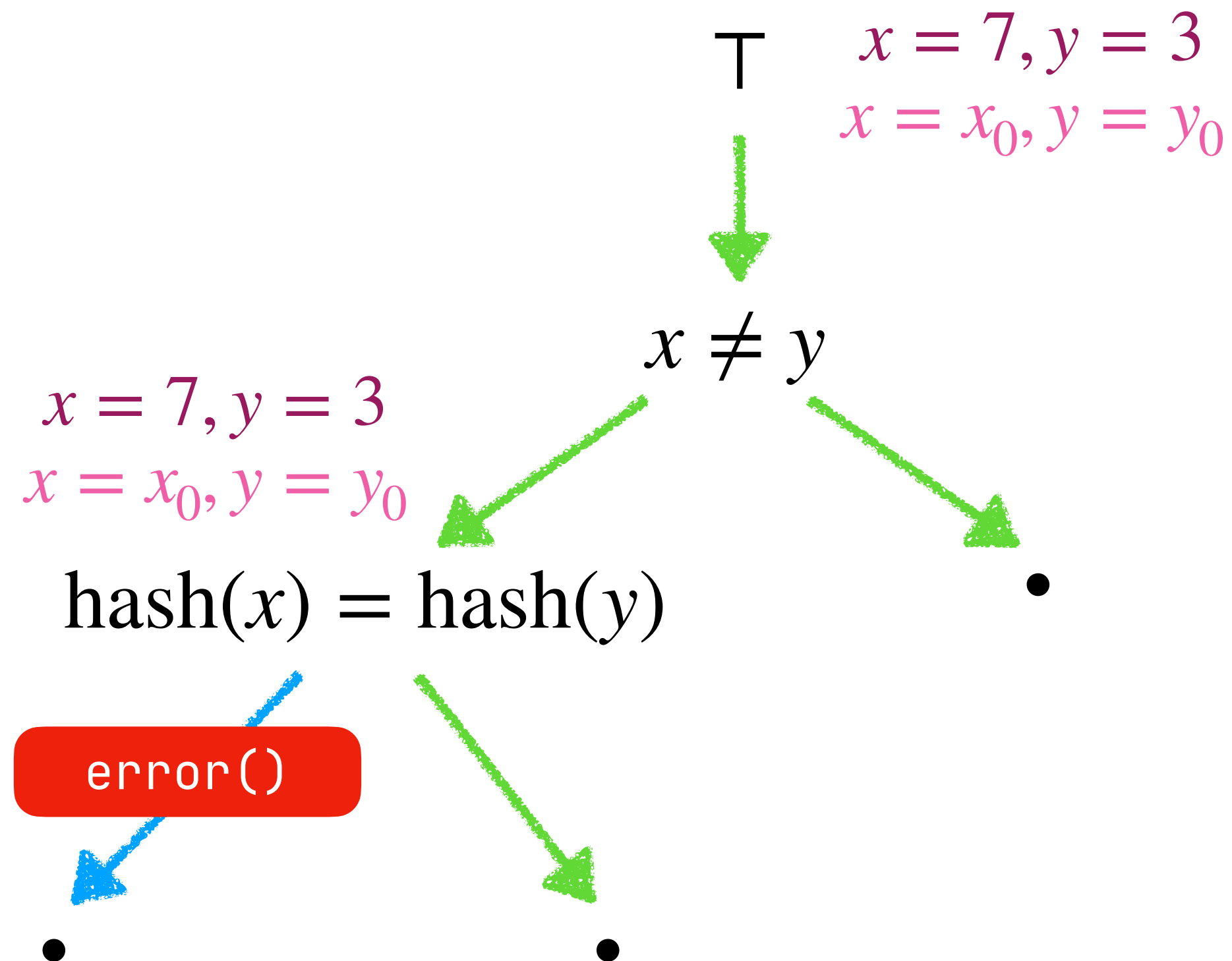


Want to explore $7 \neq y_0 \wedge \text{hash}(7) = \text{hash}(y_0)$

This is still hard to solve!

Concolic Testing: Example 3

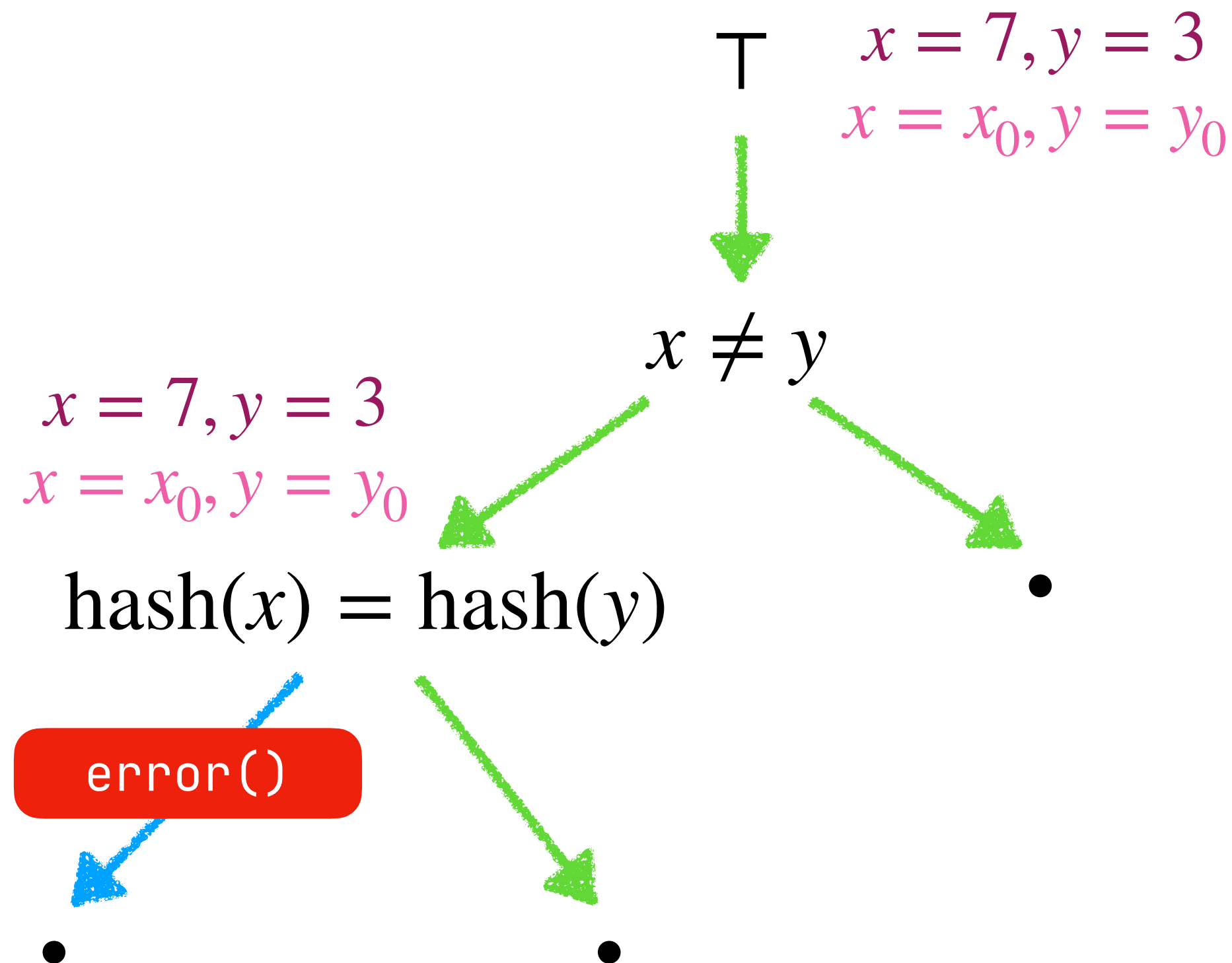
```
def bar(x, y):  
    if x  $\neq$  y:  
        if hash(x) == hash(y):  
            error()
```



Want to explore $7 \neq 3 \wedge \text{hash}(7) = \text{hash}(3)$

Concolic Testing: Example 3

```
def bar(x, y):  
    if x  $\neq$  y:  
        if hash(x) == hash(y):  
            error()
```



Want to explore $7 \neq 3 \wedge \text{hash}(7) = \text{hash}(3)$

Unsatisfiable $\implies$ The error is missed

What about loops?

What about loops?

- Infinite computation tree

What about loops?

- Infinite computation tree
- Symbolic execution: Unroll the loops boundedly

What about loops?

- Infinite computation tree
- Symbolic execution: Unroll the loops boundedly
- Concolic testing: Solve for input where the number of loop iterations satisfies some bound

Conclusion

- Symbolic execution
 - Execute on symbolic values
 - With complicated constraints or loops, can raise false alarm
- Concolic testing
 - Guided by concrete values, but use symbolic path conditions to generate new tests
 - With complicated constraints or loops, can falsely report “no bug”

References

Examples are adapted from

- Slides by Azadeh Farzan, Nick Feng, Caroline Hu, Marsha Chechik and Michael Hicks
- “Software Analysis Lectures” Ch. 11 by Mayur Naik