

# Symbolic Execution

# One Issue: Abstraction

---

- Abstraction lets us scale and model all possible runs
  - But it also introduces conservatism
  - \*-sensitivities attempt to deal with this
    - \* = flow-, context-, path-, field-, etc
  - But they are never enough
- Static analysis abstraction  $\neq$  developer abstraction
  - Because the developer didn't have them in mind

# Symbolic Execution [King, CACM 1976]

---

- Testing works
  - But, each test only explores one possible execution
    - `assert(f(3) == 5)`
  - *We hope* test cases generalize, but no guarantees
- Symbolic execution generalizes testing
  - Allows *unknown* symbolic variables in evaluation
    - `y =  $\alpha$ ; assert(f(y) == 2*y-1);`
  - If execution path depends on unknown, conceptually *fork* symbolic executor
    - `int f(int x) { if (x > 0) then return 2*x - 1; else return 10; }`

# Symbolic Execution Example

---

```
1. int a =  $\alpha$ , b =  $\beta$ , c =  $\gamma$ ;  
2.           // symbolic  
3. int x = 0, y = 0, z = 0;  
4. if (a) {  
5.   x = -2;  
6. }  
7. if (b < 5) {  
8.   if (!a && c) { y = 1; }  
9.   z = 2;  
10. }  
11. assert(x+y+z!=3)
```

# Symbolic Execution Example

---

```
1. int a =  $\alpha$ , b =  $\beta$ , c =  $\gamma$ ;  
2.           // symbolic  
3. int x = 0, y = 0, z = 0;  
4. if (a) {  
5.   x = -2;  
6. }  
7. if (b < 5) {  
8.   if (!a && c) { y = 1; }  
9.   z = 2;  
10. }  
11. assert(x+y+z!=3)
```

$x=0, y=0, z=0$

# Symbolic Execution Example

---

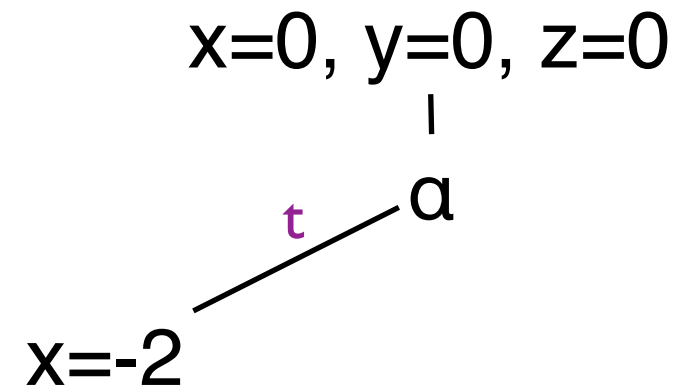
```
1. int a =  $\alpha$ , b =  $\beta$ , c =  $\gamma$ ;  
2.           // symbolic  
3. int x = 0, y = 0, z = 0;  
4. if (a) {  
5.   x = -2;  
6. }  
7. if (b < 5) {  
8.   if (!a && c) { y = 1; }  
9.   z = 2;  
10. }  
11. assert(x+y+z!=3)
```

x=0, y=0, z=0  
|  
a

# Symbolic Execution Example

---

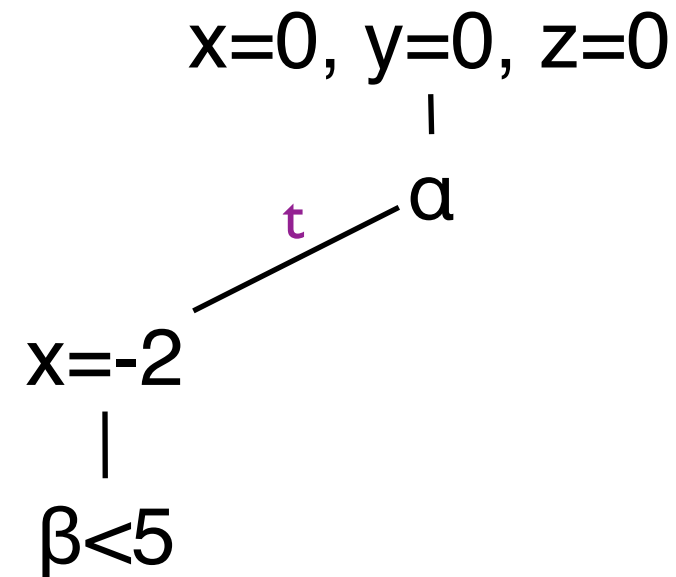
```
1. int a =  $\alpha$ , b =  $\beta$ , c =  $\gamma$ ;  
2.           // symbolic  
3. int x = 0, y = 0, z = 0;  
4. if (a) {  
5.   x = -2;  
6. }  
7. if (b < 5) {  
8.   if (!a && c) { y = 1; }  
9.   z = 2;  
10. }  
11. assert(x+y+z!=3)
```



# Symbolic Execution Example

---

```
1. int a =  $\alpha$ , b =  $\beta$ , c =  $\gamma$ ;  
2.           // symbolic  
3. int x = 0, y = 0, z = 0;  
4. if (a) {  
5.   x = -2;  
6. }  
7. if (b < 5) {  
8.   if (!a && c) { y = 1; }  
9.   z = 2;  
10. }  
11. assert(x+y+z!=3)
```

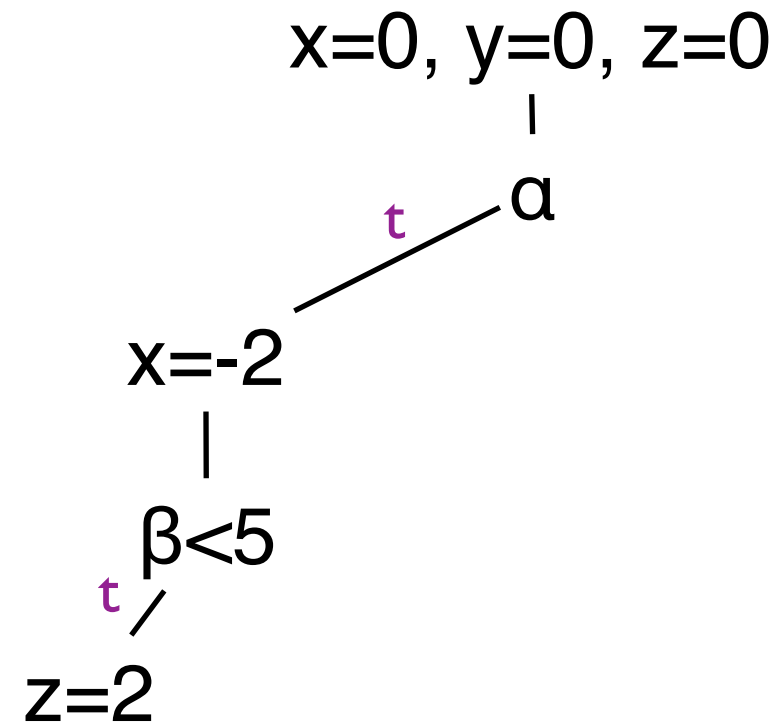




# Symbolic Execution Example

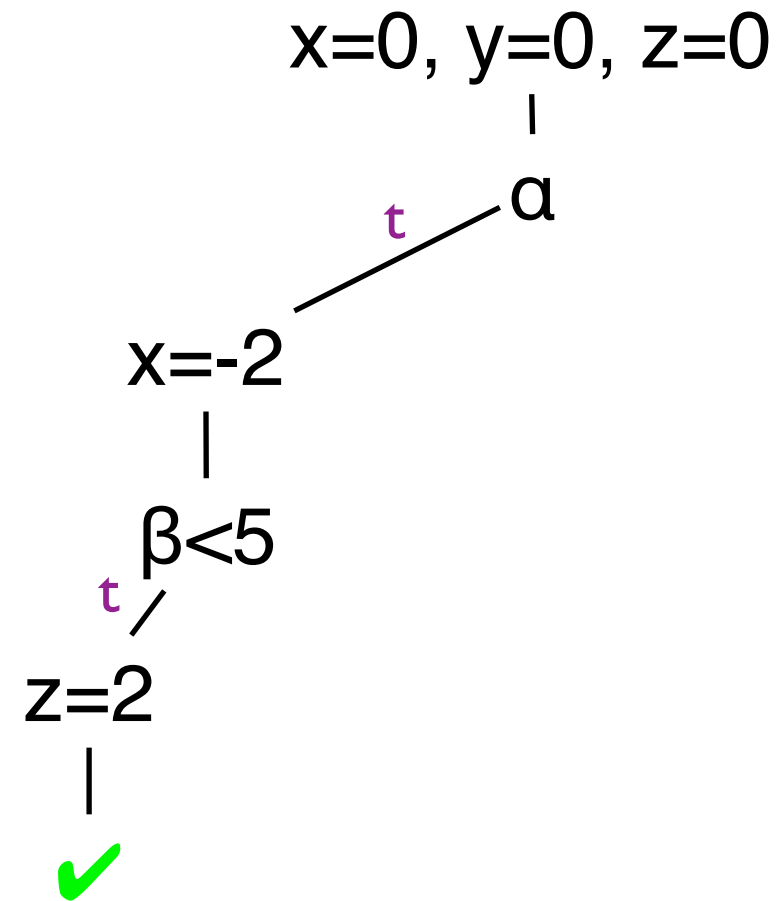
---

```
1. int a =  $\alpha$ , b =  $\beta$ , c =  $\gamma$ ;  
2.           // symbolic  
3. int x = 0, y = 0, z = 0;  
4. if (a) {  
5.   x = -2;  
6. }  
7. if (b < 5) {  
8.   if (!a && c) { y = 1; }  
9.   z = 2;  
10. }  
11. assert(x+y+z!=3)
```



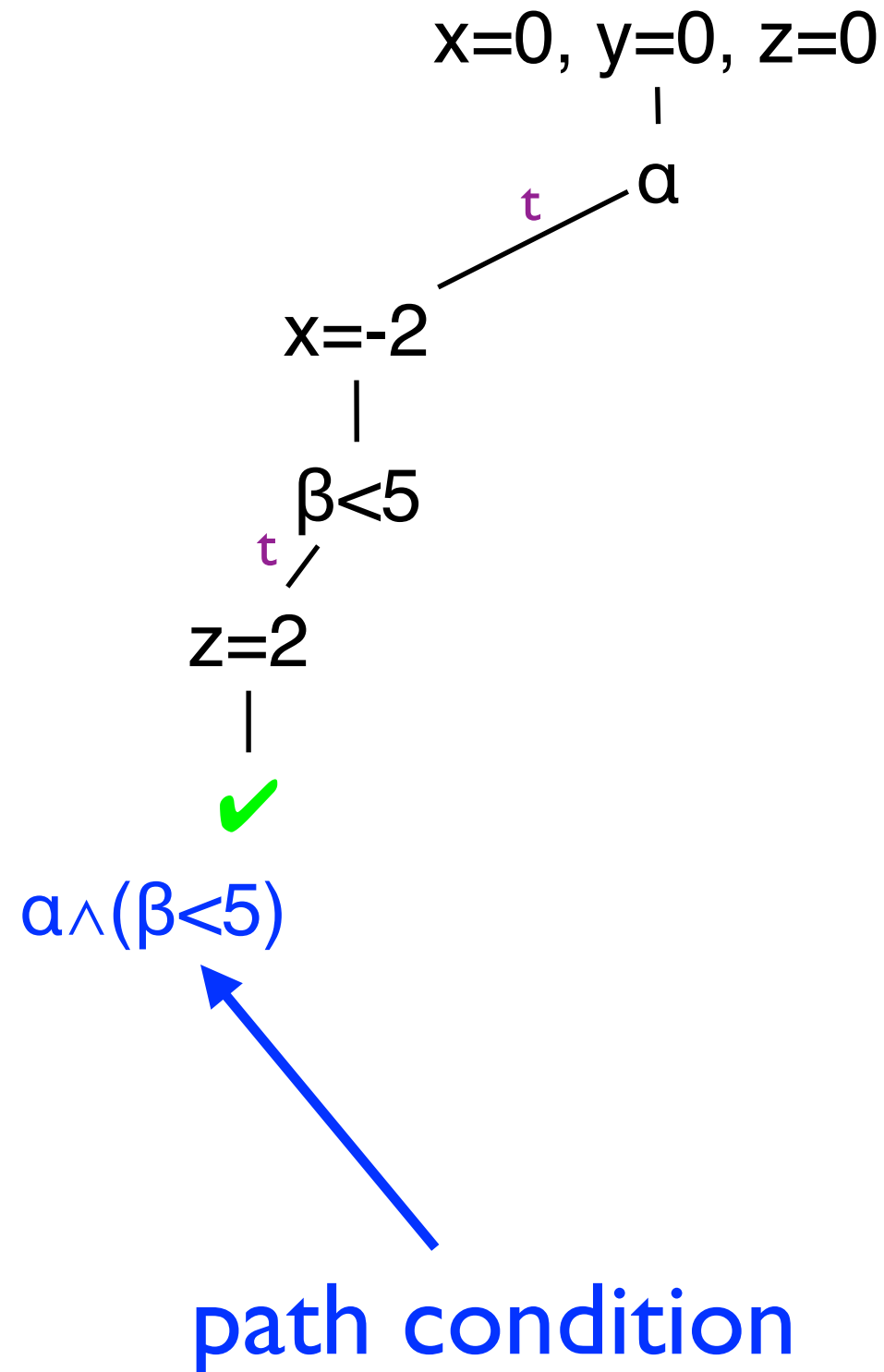
# Symbolic Execution Example

```
1. int a =  $\alpha$ , b =  $\beta$ , c =  $\gamma$ ;  
2.           // symbolic  
3. int x = 0, y = 0, z = 0;  
4. if (a) {  
5.   x = -2;  
6. }  
7. if (b < 5) {  
8.   if (!a && c) { y = 1; }  
9.   z = 2;  
10. }  
11. assert(x+y+z!=3)
```



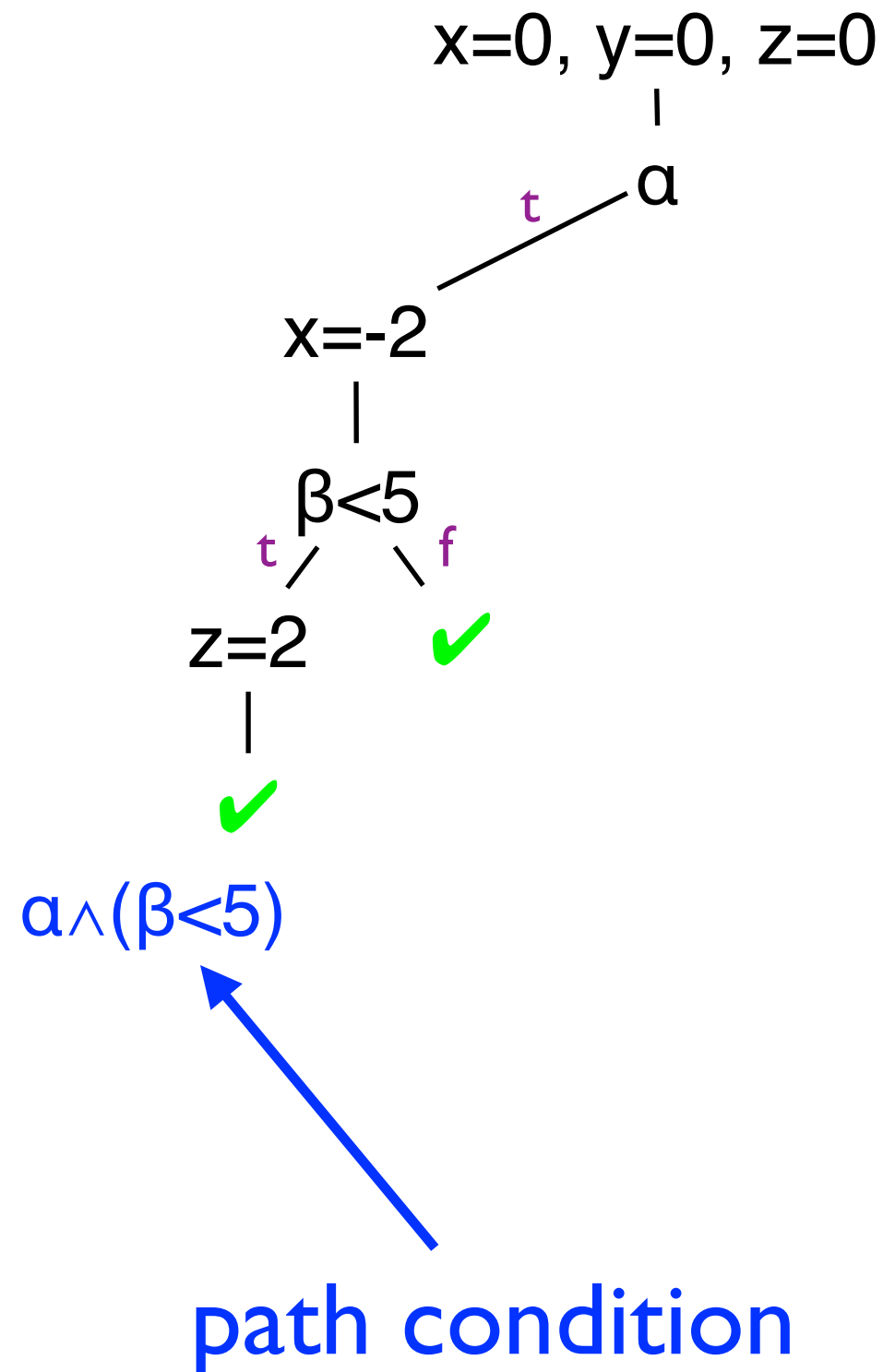
# Symbolic Execution Example

```
1. int a =  $\alpha$ , b =  $\beta$ , c =  $\gamma$ ;  
2.           // symbolic  
3. int x = 0, y = 0, z = 0;  
4. if (a) {  
5.   x = -2;  
6. }  
7. if (b < 5) {  
8.   if (!a && c) { y = 1; }  
9.   z = 2;  
10. }  
11. assert(x+y+z!=3)
```



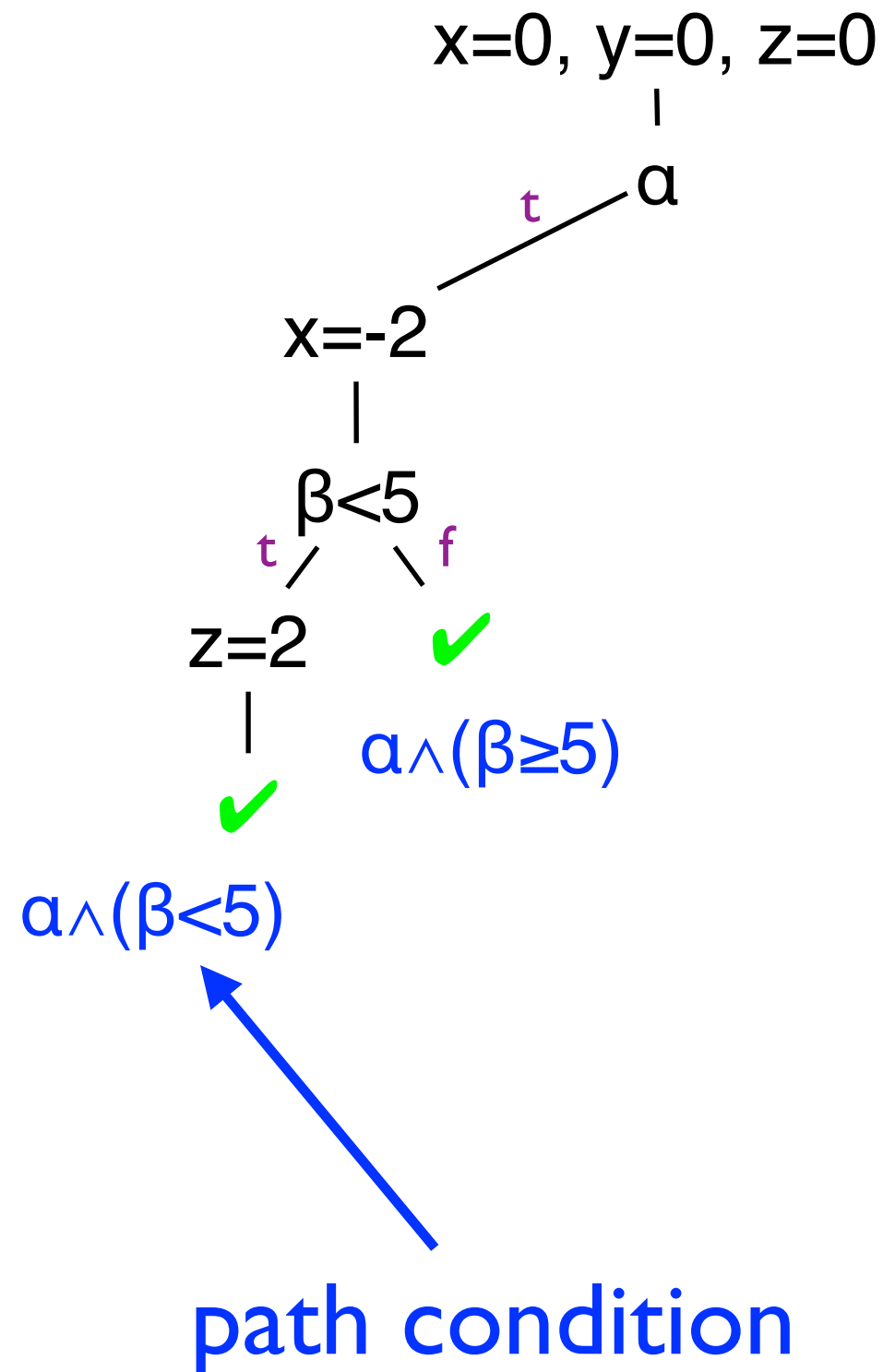
# Symbolic Execution Example

```
1. int a =  $\alpha$ , b =  $\beta$ , c =  $\gamma$ ;  
2.           // symbolic  
3. int x = 0, y = 0, z = 0;  
4. if (a) {  
5.   x = -2;  
6. }  
7. if (b < 5) {  
8.   if (!a && c) { y = 1; }  
9.   z = 2;  
10. }  
11. assert(x+y+z!=3)
```



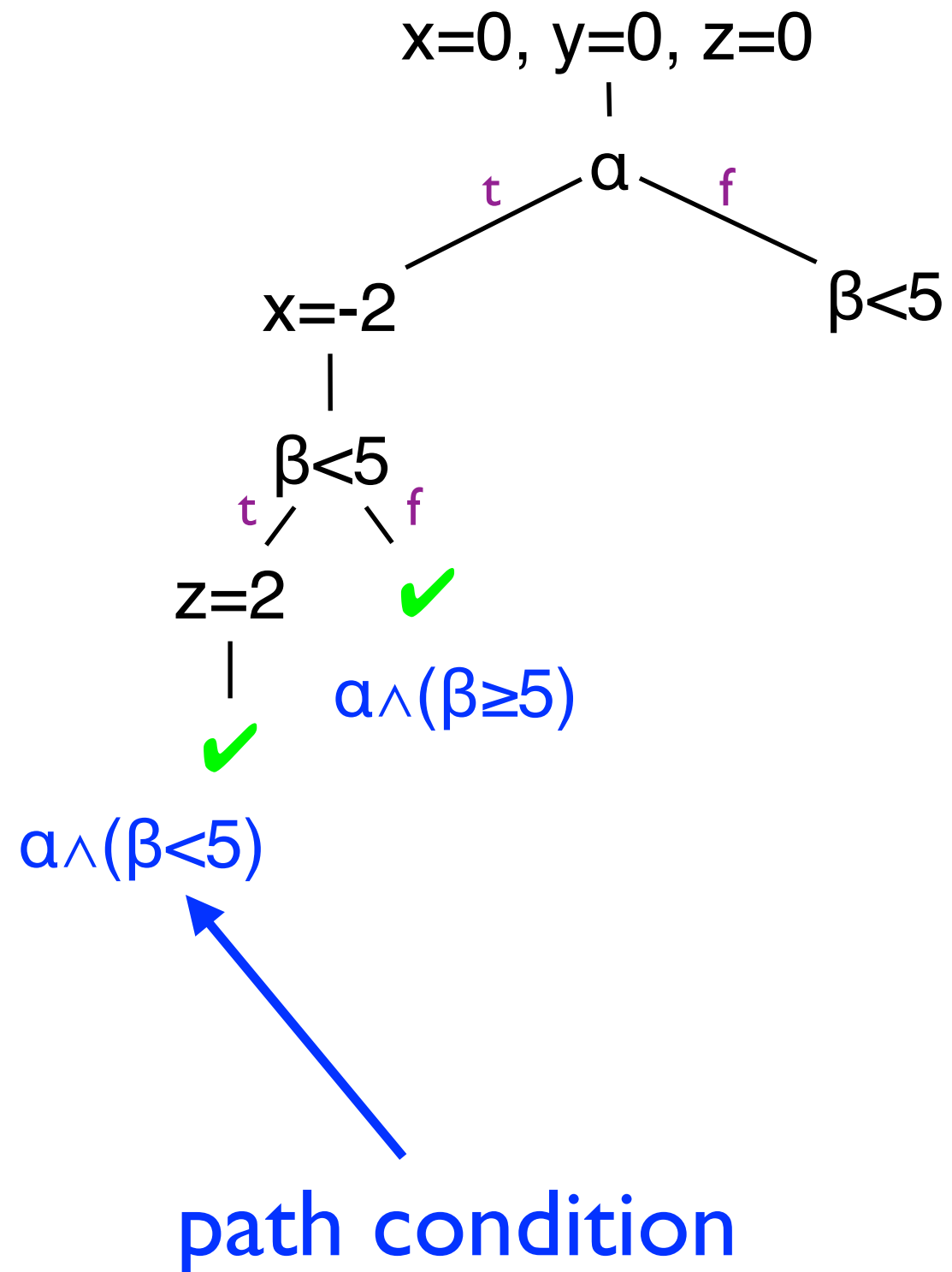
# Symbolic Execution Example

```
1. int a =  $\alpha$ , b =  $\beta$ , c =  $\gamma$ ;  
2.           // symbolic  
3. int x = 0, y = 0, z = 0;  
4. if (a) {  
5.   x = -2;  
6. }  
7. if (b < 5) {  
8.   if (!a && c) { y = 1; }  
9.   z = 2;  
10. }  
11. assert(x+y+z!=3)
```



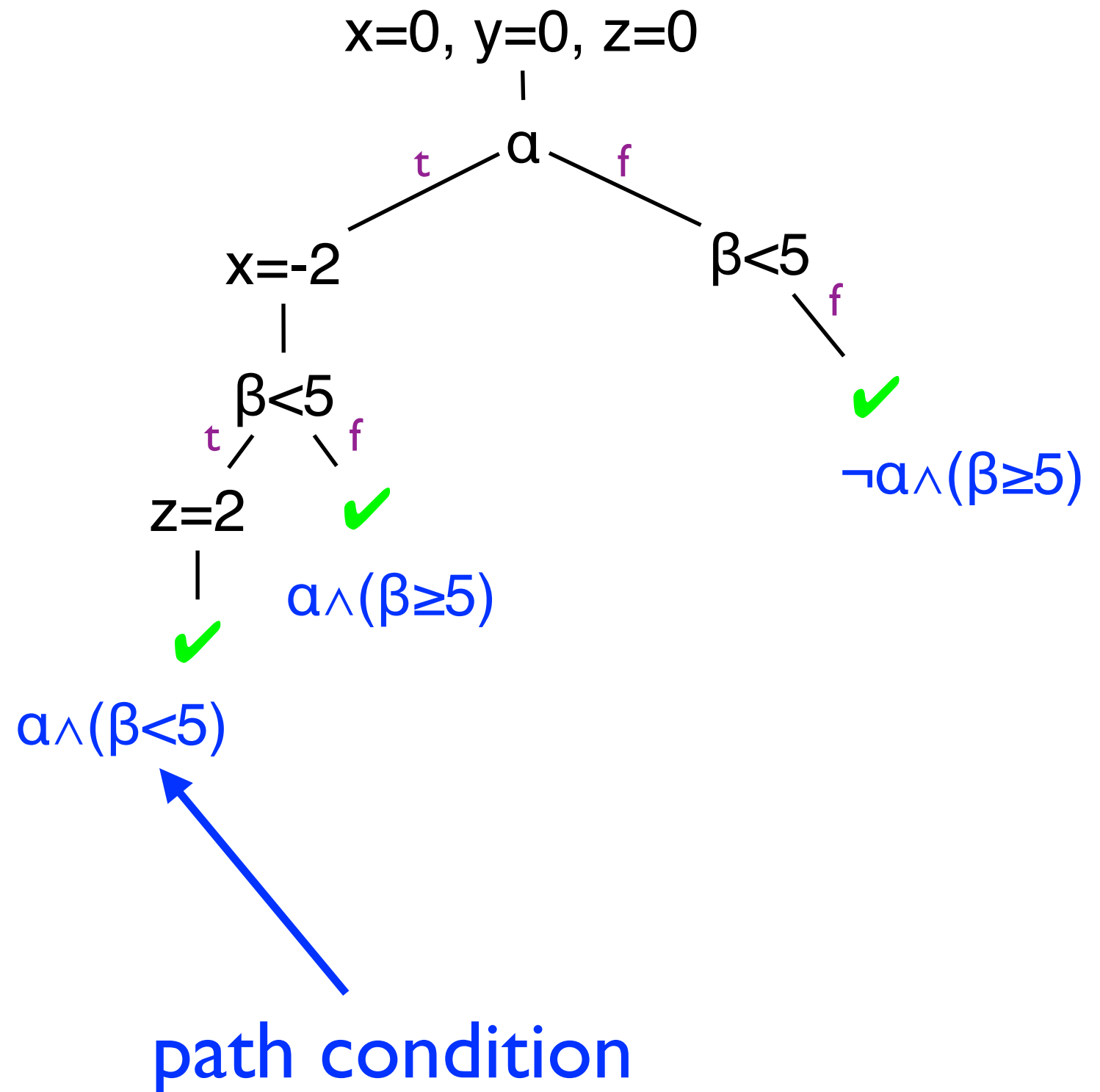
# Symbolic Execution Example

```
1. int a =  $\alpha$ , b =  $\beta$ , c =  $\gamma$ ;  
2.           // symbolic  
3. int x = 0, y = 0, z = 0;  
4. if (a) {  
5.   x = -2;  
6. }  
7. if (b < 5) {  
8.   if (!a && c) { y = 1; }  
9.   z = 2;  
10. }  
11. assert(x+y+z!=3)
```



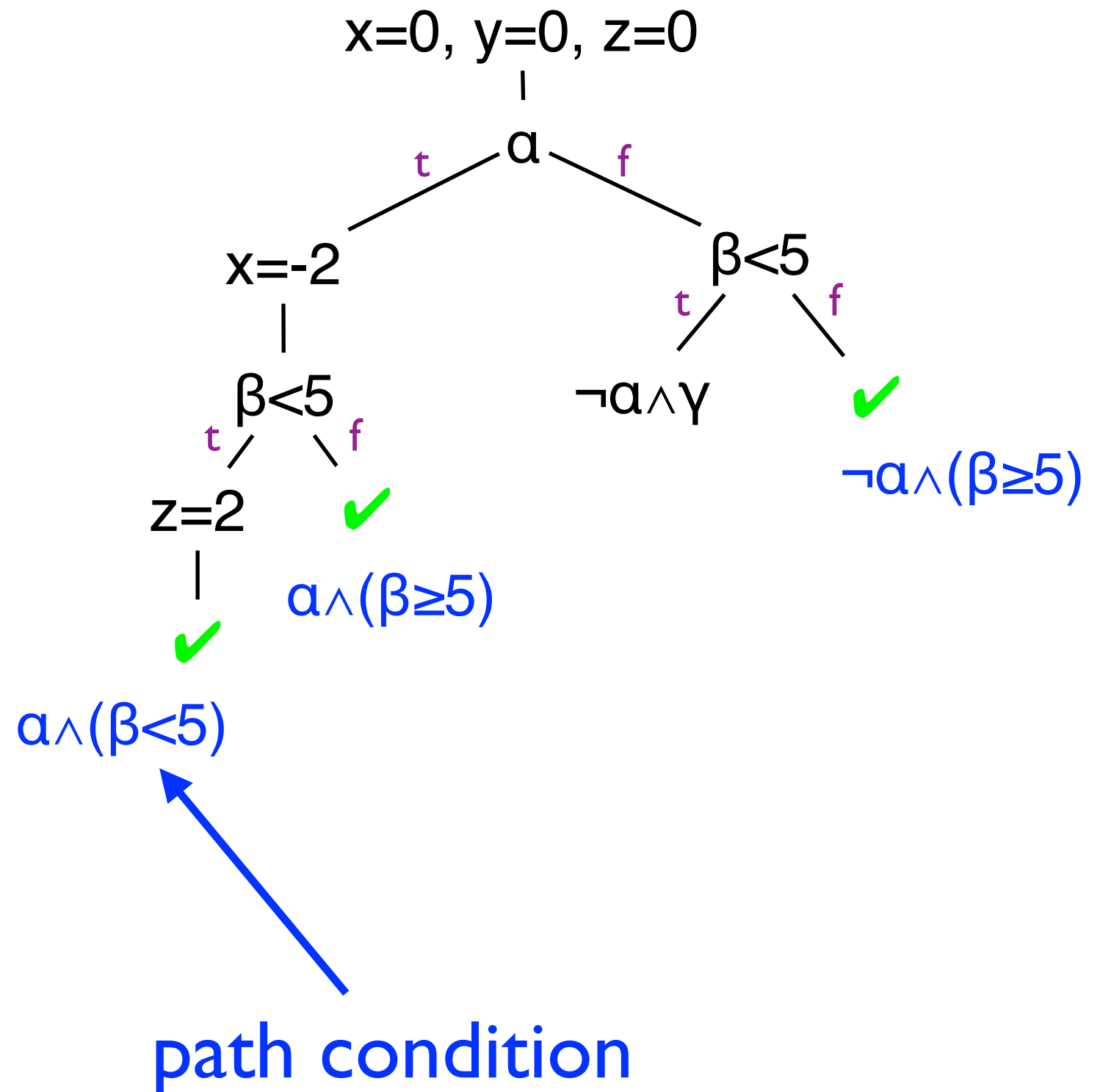
# Symbolic Execution Example

```
1. int a =  $\alpha$ , b =  $\beta$ , c =  $\gamma$ ;  
2.           // symbolic  
3. int x = 0, y = 0, z = 0;  
4. if (a) {  
5.   x = -2;  
6. }  
7. if (b < 5) {  
8.   if (!a && c) { y = 1; }  
9.   z = 2;  
10. }  
11. assert(x+y+z!=3)
```



# Symbolic Execution Example

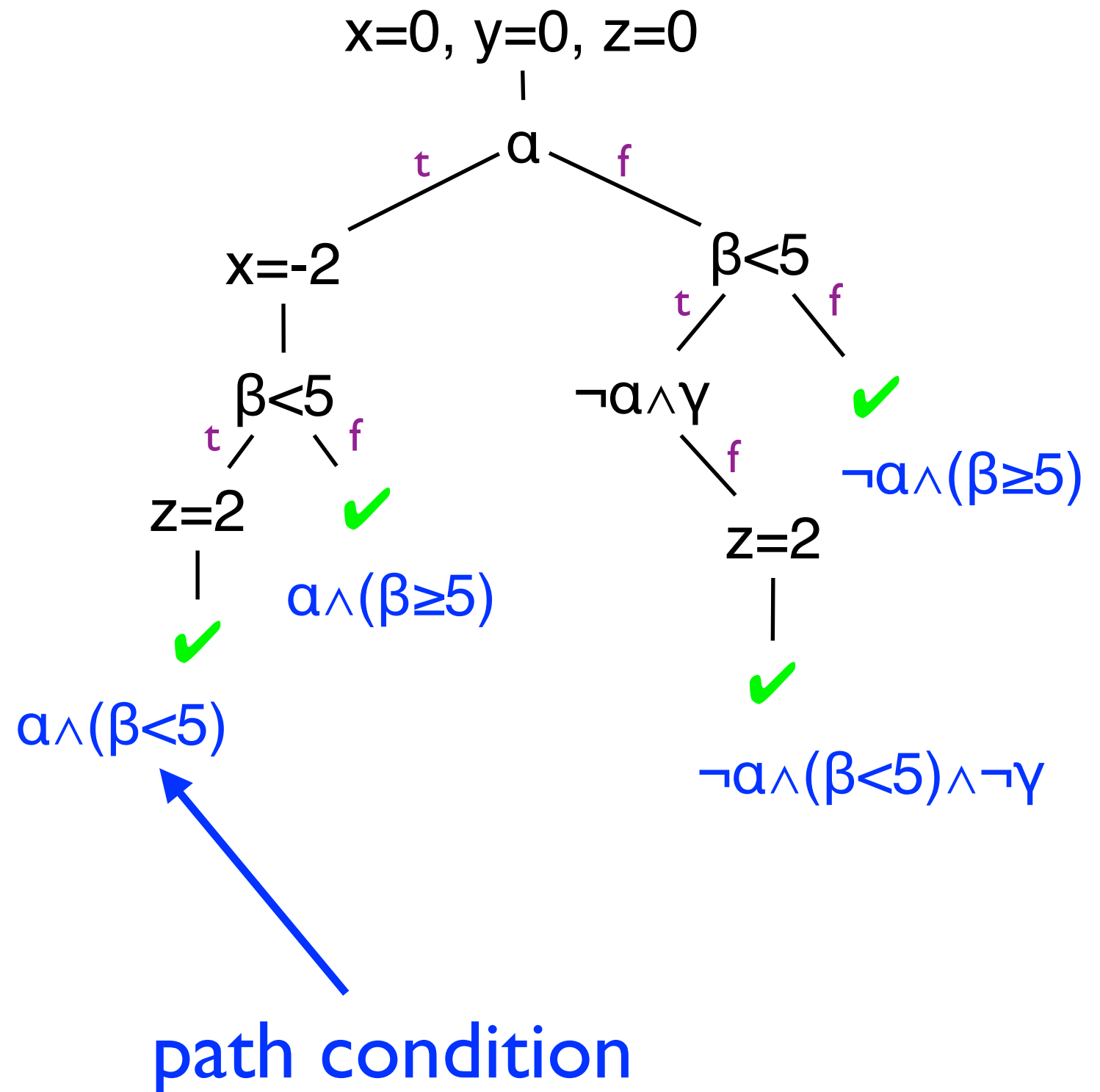
```
1. int a =  $\alpha$ , b =  $\beta$ , c =  $\gamma$ ;  
2.           // symbolic  
3. int x = 0, y = 0, z = 0;  
4. if (a) {  
5.   x = -2;  
6. }  
7. if (b < 5) {  
8.   if (!a && c) { y = 1; }  
9.   z = 2;  
10. }  
11. assert(x+y+z!=3)
```





# Symbolic Execution Example

```
1. int a =  $\alpha$ , b =  $\beta$ , c =  $\gamma$ ;  
2.           // symbolic  
3. int x = 0, y = 0, z = 0;  
4. if (a) {  
5.   x = -2;  
6. }  
7. if (b < 5) {  
8.   if (!a && c) { y = 1; }  
9.   z = 2;  
10. }  
11. assert(x+y+z!=3)
```

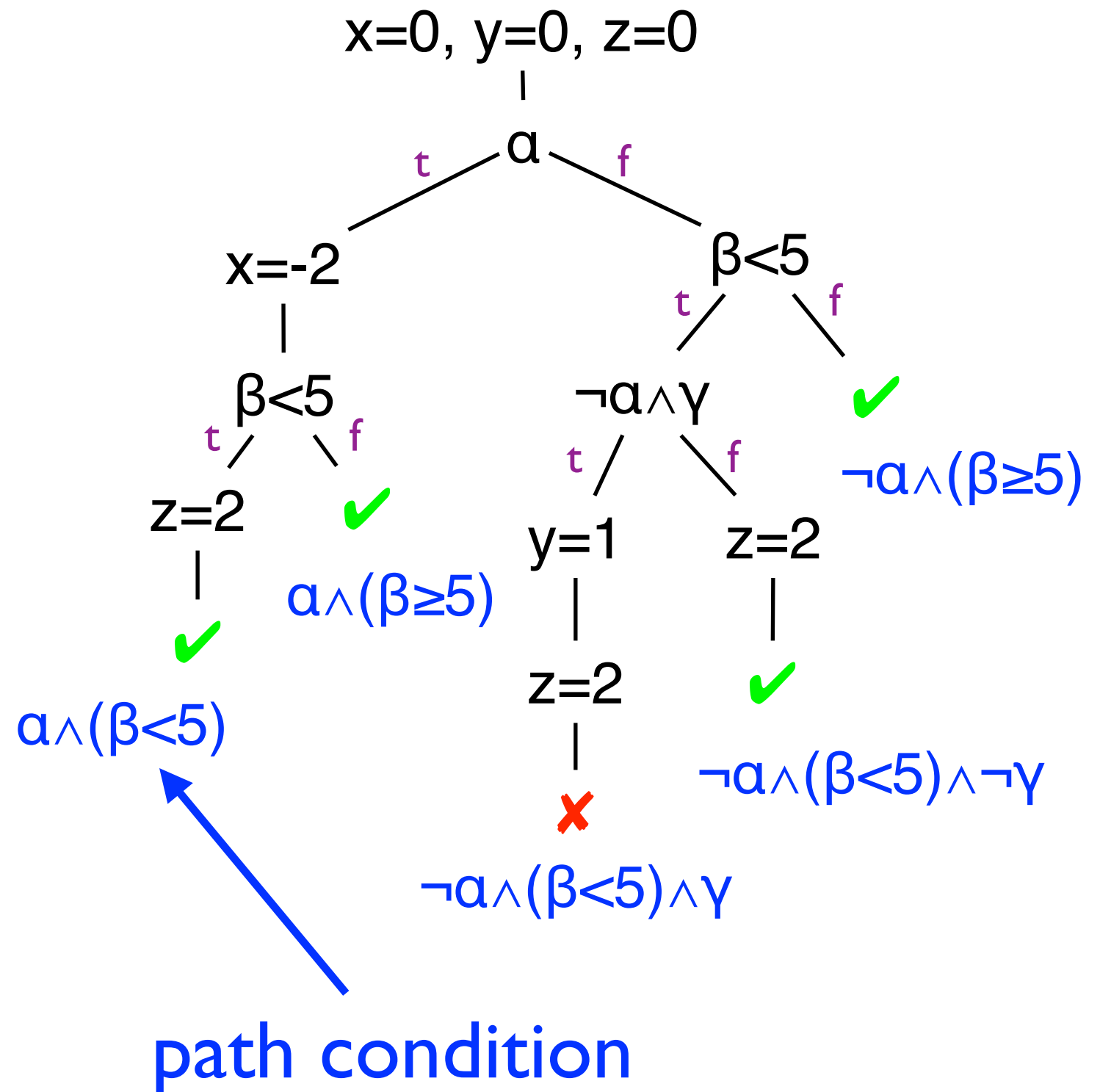


# Symbolic Execution Example

```

1. int a = a, b = β, c = γ;
2.           // symbolic
3. int x = 0, y = 0, z = 0;
4. if (a) {
5.   x = -2;
6. }
7. if (b < 5) {
8.   if (!a && c) { y = 1; }
9.   z = 2;
10.}
11. assert(x+y+z!=3)

```



# Why Is This Possible?

---

- There are very powerful SMT/SAT solvers today
  - SMT = Satisfiability Modulo Theories = SAT++
  - Can solve very large instances, very quickly
    - Lets us check assertions, prune infeasible paths
  - We've used Z3, STP, and Yices
- Recent success: bug finding
  - Heuristic search through space of possible executions
  - Find really interesting bugs