EXERCISE 28

FUZZING REVIEW

Give 2 example programs, each with 1 if statement. One of the programs should be likely for a fuzzer to generate full line coverage, the other should be difficult for the fuzzer to generate full line coverage.

```
#include "stdio.h"
int main() {
    int a;
    a = getchar();
    if (a >0) {
        return 1/0;
    }
```

```
#include "stdio.h"
int main() {
    int a;
    a = getchar();
    if (a == ~~12345~~ 12) {
        return 1/0;
    }
}
```

EXERCISE 28 SOLUTION

FUZZING REVIEW

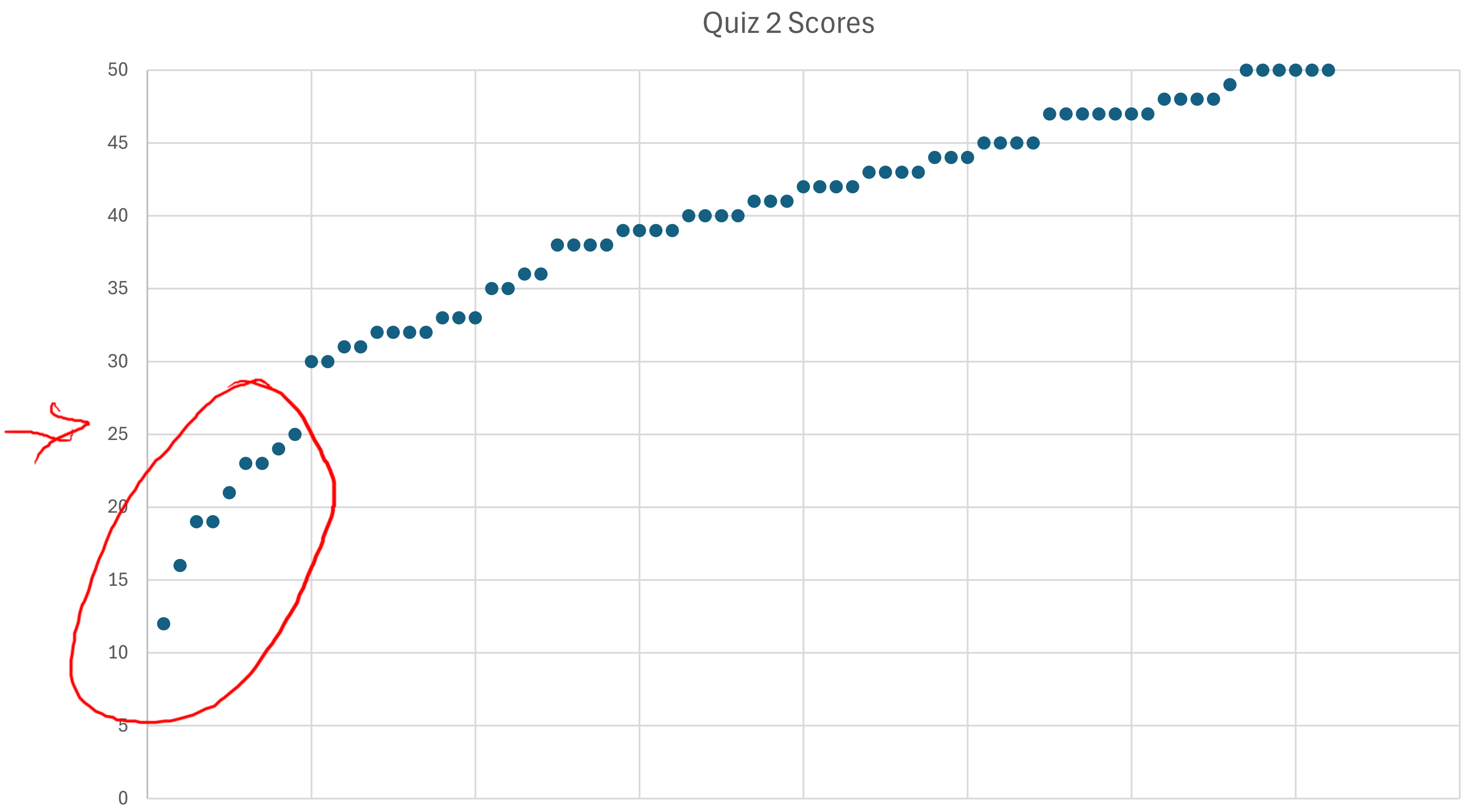
Give 2 example programs, each with 1 if statement. One of the programs should be likely for a fuzzer to generate full line coverage, the other should be difficult for the fuzzer to generate full line coverage.

Quiz 2

ADMINISTRIVIA AND ANNOUNCEMENTS

QUIZ 2

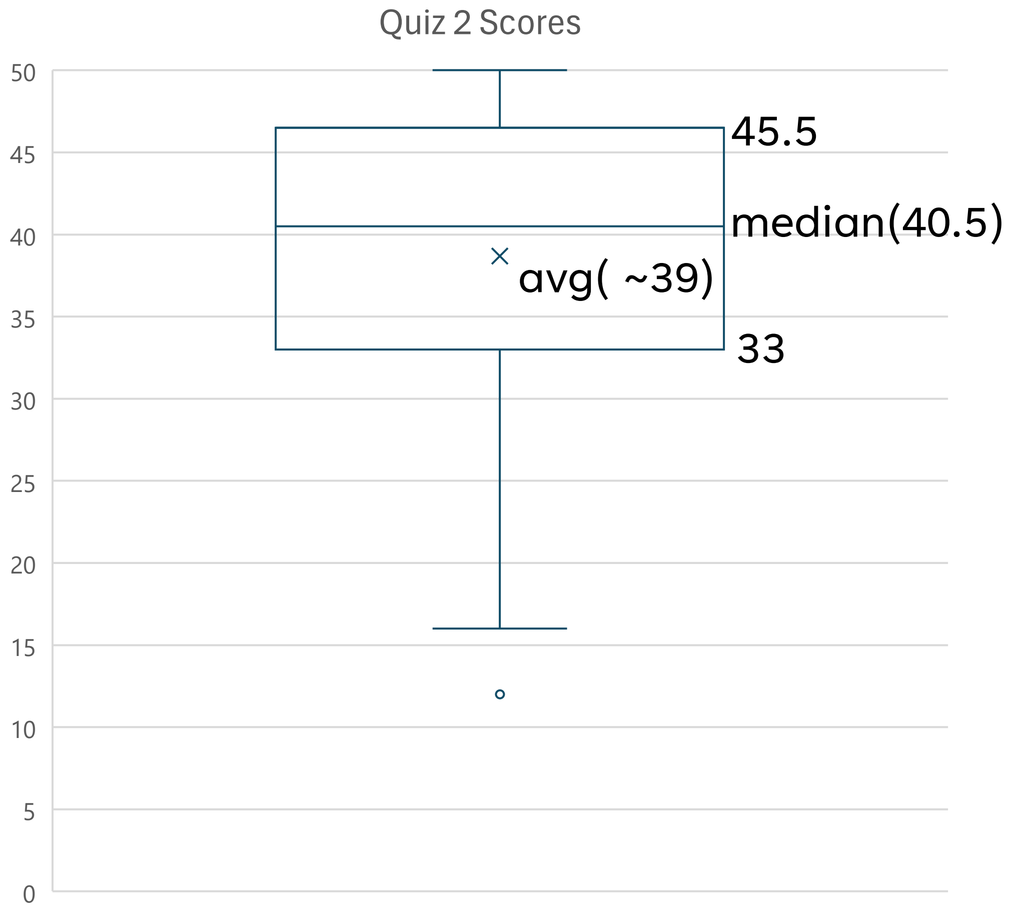
ADMINISTRIVIA



QUIZ 2

ADMINISTRIVIA

P	~8%
A	~22%
B	~25%
C	~17%
D	~15%
F	~13%



SYMBOLIC EXECUTION

EECS 677: Software Security Evaluation

Drew Davidson

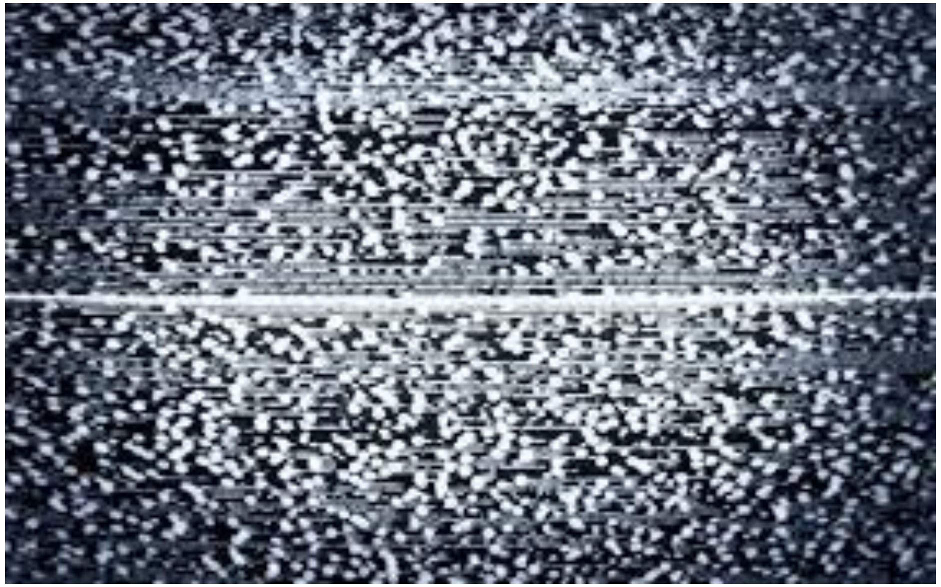
PREVIOUSLY: FUZZING

OUTLINE / OVERVIEW

Generating Random test cases

Surprisingly effective in practice

Main challenge is exploring "new" behavior



The random "fuzz" of white noise

RESEARCH DIRECTION: “GUNKING”

FUZZING

Fuzzing as adversarial recon

Fuzzing is so good at finding bugs that even the bad guys do it

Perhaps a program should deploy Anti-Fuzzing tech

What would that look like?

THE PROBLEM OF COVERAGE: STATIC

OUTLINE / OVERVIEW

This program is well-analyzed in the abstract domain of signs

```
#include "stdlib.h"
int main(){
    int c = getchar();
    if (c == 0 && c == 1) {
        return 1 / 0;
    }
}
```

This program has an FP in the abstract domain of signs

```
#include "stdlib.h"
int main(){
    int c = getchar();
    if (c == 1 && c == 2) {
        return 1 / 0;
    }
}
```

THE PROBLEM OF COVERAGE: DYNAMIC

SYMBOLIC EXECUTION

```c
#include "stdlib.h"
int main(){
    int c = getchar();
    if (c == 12345) {
        return 1 / 0;
    }
}
```

WHAT MATTERS IS PREDICATES

SYMBOLIC EXECUTION

```
#include "stdlib.h"
int main(){
    int c = getchar();
    if ~~(c == 12345~~) {
        return 1 / 0;
    }
}
```

c > 123 && c < 135

PREDICATES GET IN THE WAY!

SYMBOLIC EXECUTION

```
#include "stdlib.h"
int main(){
    int c = getchar();
    if (c == 12345) {
        c = getchar();
        if (c  % 2 == 0 ) {
            return 1 / 0;
        }
    }
}
```

SYMBOLIC EXECUTION: INTUITION

SYMBOLIC EXECUTION

$\boxed{\text{madeSubway} = \alpha}$

if(madeSubway)

madeSubway =

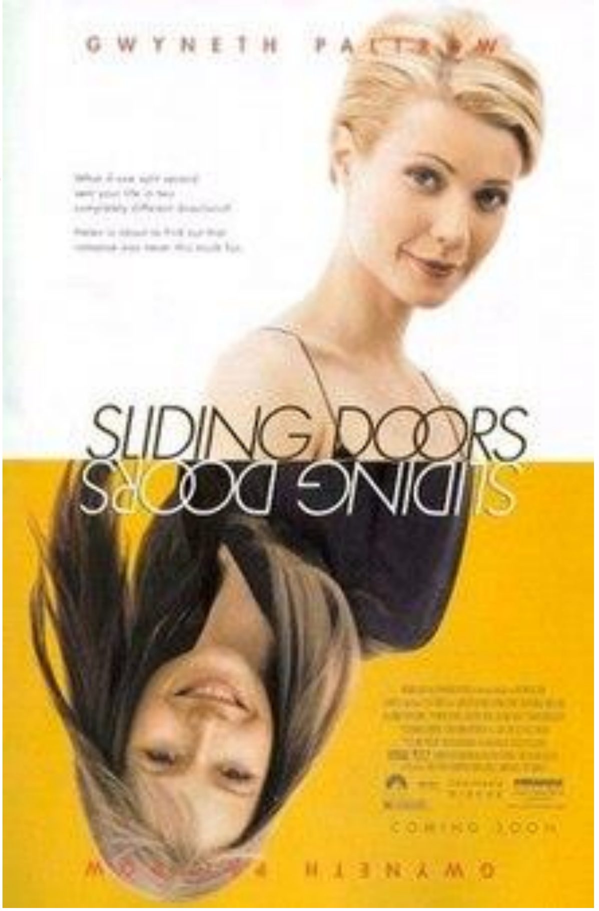
α

$\wedge$

$\alpha == \text{true}$

madeSubway =

$\alpha \overset{\wedge}{=}= \text{false}$



EXPLORE BOTH SIDES OF PREDICATE!

SYMBOLIC EXECUTION: INTUITION

```
 1: #include "stdlib.h"
 2: int main(){
 3:     int c = getchar();
 4:     if (c == 12345) {
 5:         ~~c = getchar();~~
 6:         if (c  % 2 == 0 ) {
 7:             return 1 / 0;
 8:         }
 9:     }
10: }
```

$c == \gamma \wedge \gamma == 12345 \wedge \gamma \% 2 == 0$

$2: r = \alpha$

$3: c = \gamma$

$4a: c = \gamma \wedge \gamma == 12345$

$4b: c = \gamma \wedge \gamma != 12345$

$5a: c = \beta \wedge \gamma != 12345$

$9b: c = \gamma \wedge \gamma != 12345$

$6a: c = \beta \wedge \gamma == 12345 \wedge \beta \% 2 == 0$

$6c: c = \beta \wedge \gamma == 12345 \wedge \beta \% 2 != 0$

$7a$ = BUG!

THE SYMBOLIC EXECUTION TREE

SYMBOLIC EXECUTION

At each line of the program:

- Advance the symbolic program state
- Split the symbolic state into 2 versions
  1) Satisfies the branch predicate
  2) Does not satisfy the branch predicate

```c
#include "stdlib.h"
int main(){
    int c = getchar();
    if (c == 12345) {
        c = getchar();
        if (c  % 2 == 0 ) {
            return 1 / 0;
        }
    }
}
```

ELIMINATING INFEASIBLE PATHS

SYMBOLIC EXECUTION

```
#include "stdlib.h"
int main(){
    int c = getchar();
    if (c == 12345) {
        ~~c = getchar();~~
        if (c  % 2 == 0 ) {
            return 1 / 0;
        }
    }
}
```

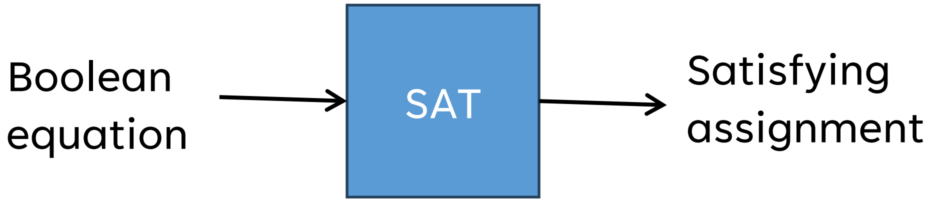
THE MAGIC OF THE SOLVER

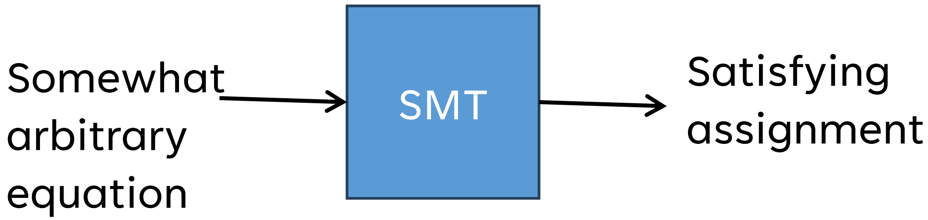
SYMBOLIC EXECUTION

$a \land \neg a \land b \land c \land \neg d \land e$

$a \land \neg a$

$a \land b$





THE SYMBOLIC EXECUTION TREE

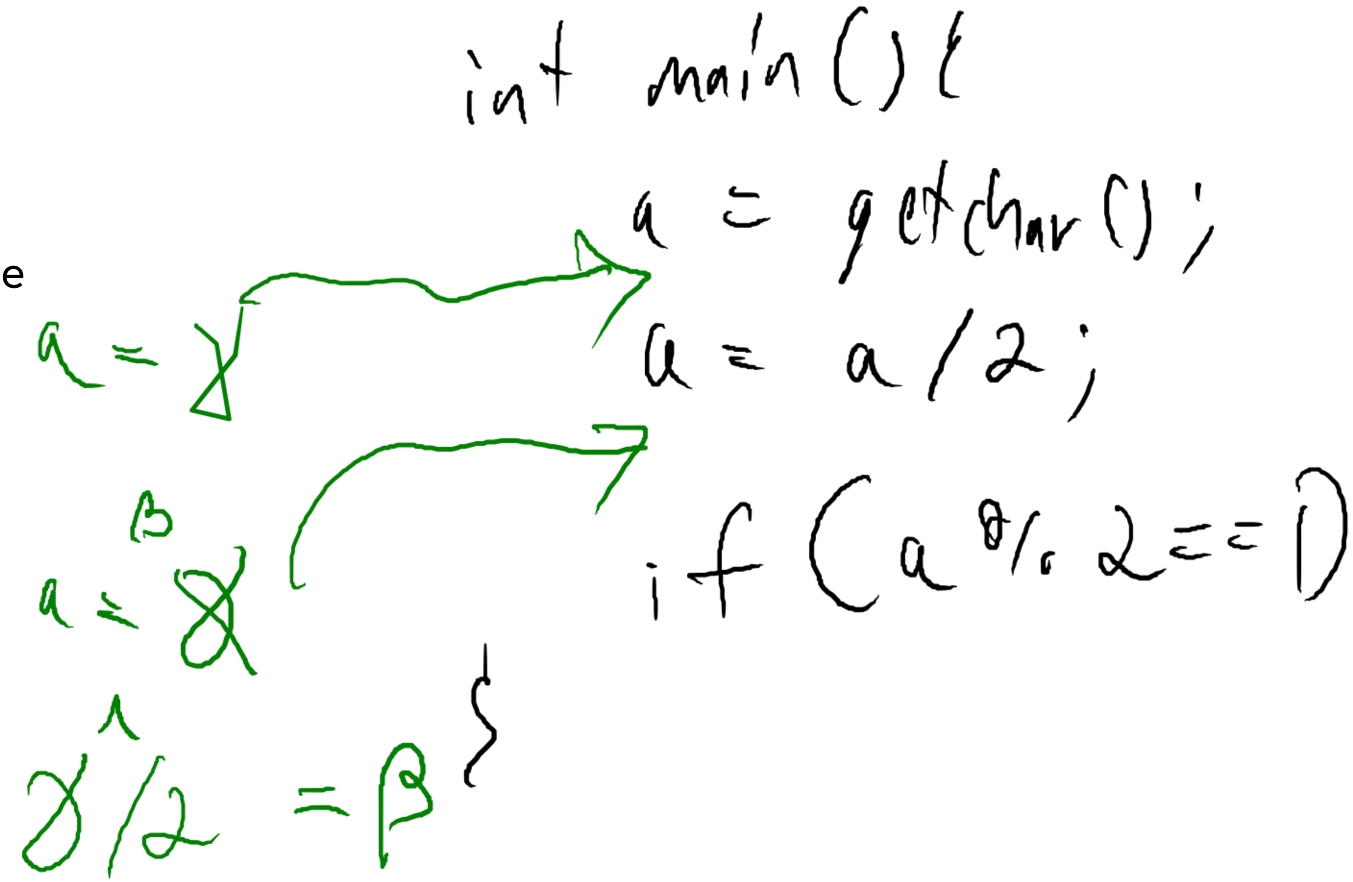
SYMBOLIC EXECUTION

At each line of the program:

- Advance the symbolic program state
- Split the symbolic state into 2 versions
  1) Satisfies the branch predicate
  2) Does not satisfy the branch predicate

  ENSURE FEASIBILITY

```c
#include "stdlib.h"
int main(){
    int c = getchar();
    if (c == 12345) {
        c = getchar();
        if (c  % 2 == 0 ) {
            return 1 / 0;
        }
    }
}
```



```
int main(){
    a = getchar();
    a = a/2;
    if(a%2==1)
}
```

$a = \gamma$

$a = \overset{\beta}{\gamma}$

$\wedge$

$\gamma/2 = \beta$

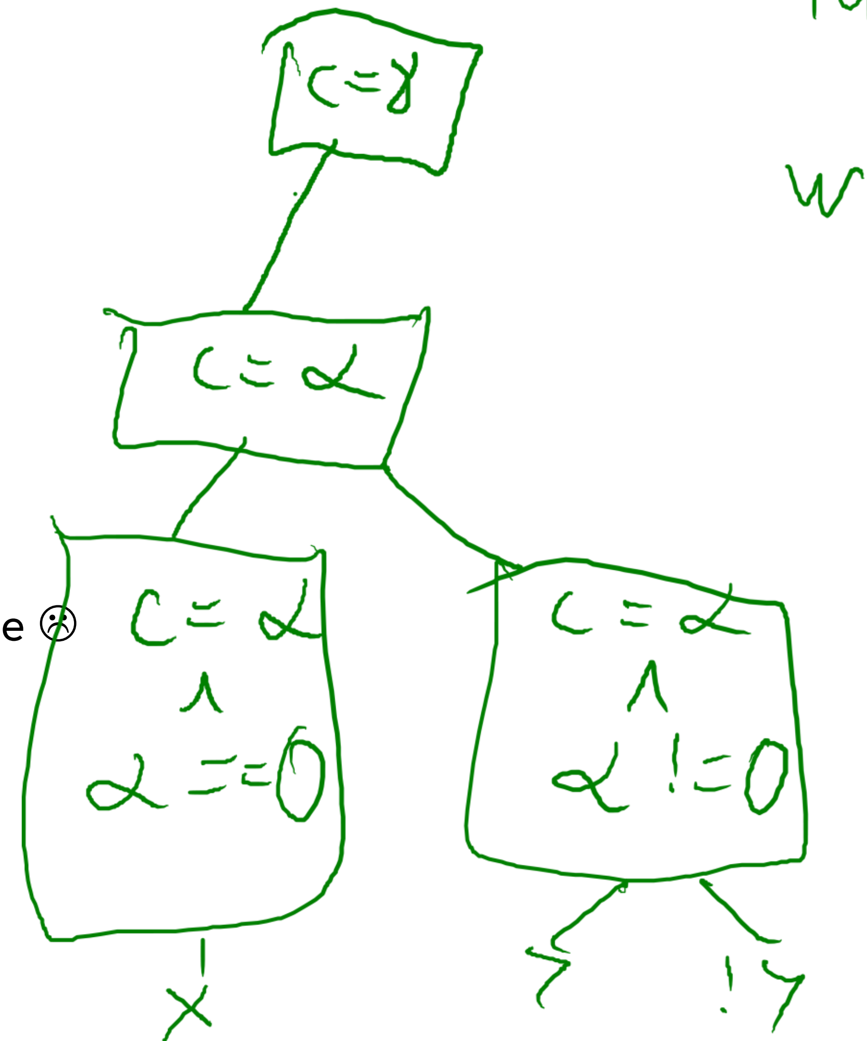
SOUNDNESS / COMPLETENESS

SYMBOLIC EXECUTION

Sound!

Complete!

May not terminate ☹



```
int c;
while (c = getchar()) {
    if (c == 7) {
        return 1/0;
    }
}
```

WRAP-UP

SYMBOLIC EXECUTION

Take all paths, don’t commit to values