EXERCISE 29

DYNAMIC ANALYSIS REVIEW

Write your name and answer the following on a piece of paper

What is the difference between unit testing and application testing?

EXERCISE 29 SOLUTION

DYNAMIC ANALYSIS REVIEW

EXERCISE 29

DYNAMIC ANALYSIS REVIEW

Write your name and answer the following on a piece of paper

What is the difference between unit testing and application testing?

EXERCISE 29 SOLUTION

DYNAMIC ANALYSIS REVIEW

EXERCISE 29

FUZZING REVIEW

Write your name and answer the following on a piece of paper

In fuzzing, it is easy to generate additional test cases for an analysis target. What are some of the strategies for ***prioritizing*** *which test case to run next?*

ADMINISTRIVIA AND ANNOUNCEMENTS

SYMBOLIC EXECUTION

EECS 677: Software Security Evaluation

Drew Davidson

WHERE WE’RE AT

DYNAMIC ANALYSIS

Generating test cases

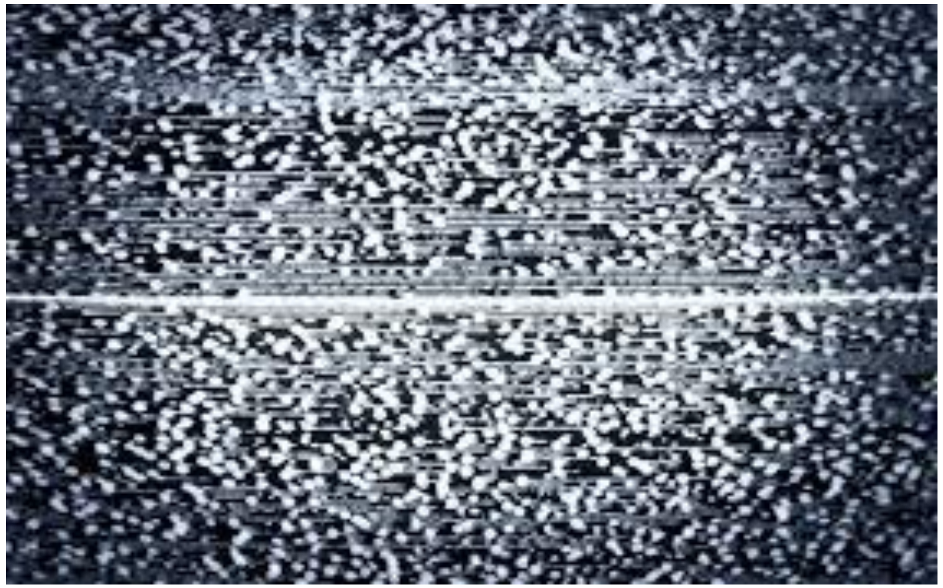
PREVIOUSLY: FUZZING

OUTLINE / OVERVIEW

Generating Random test cases

Surprisingly effective in practice

Main challenge is exploring “new” behavior



The random “fuzz” of white noise

THIS LECTURE: SYMBOLIC EXECUTION

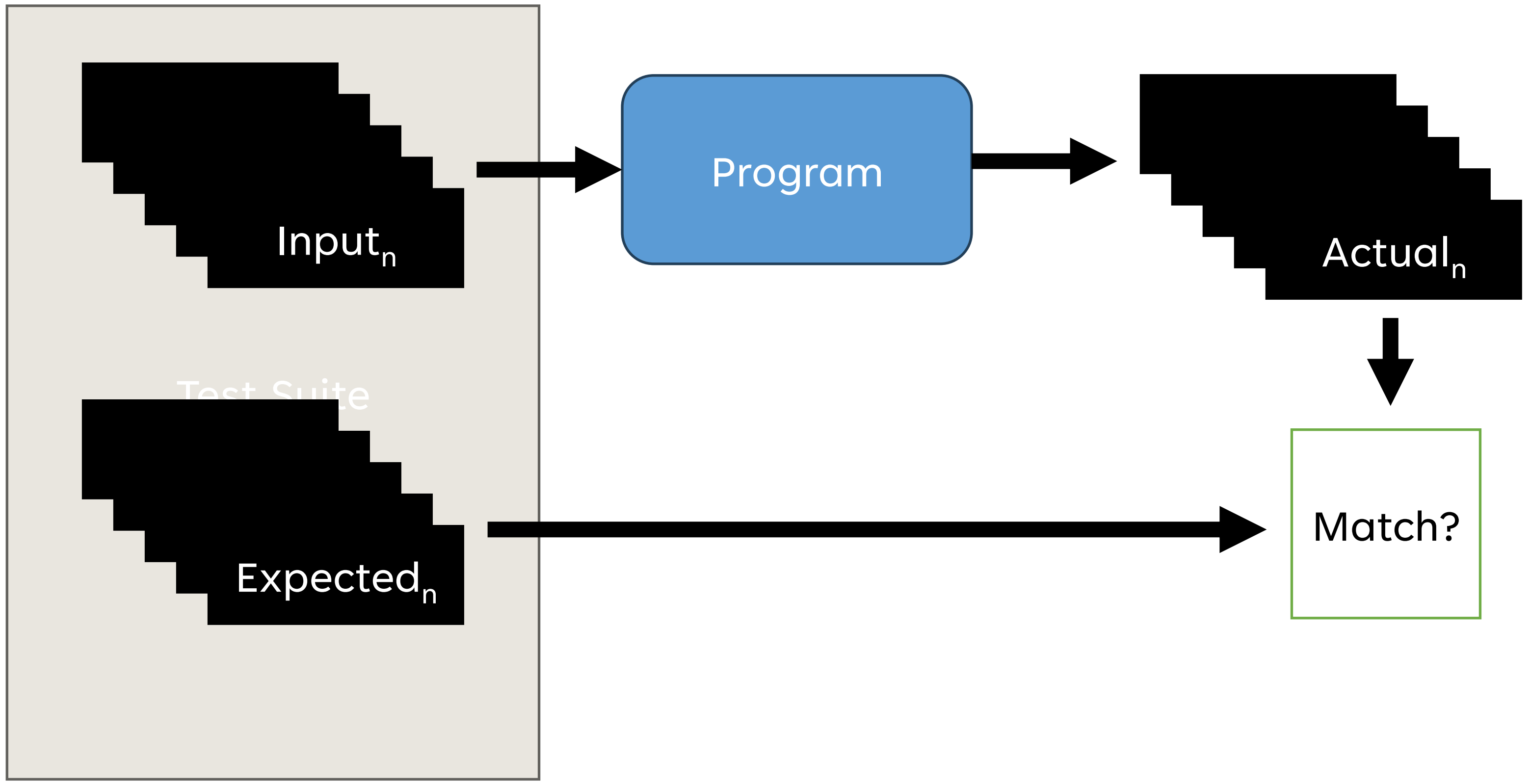
OUTLINE / OVERVIEW

A METHODICAL APPROACH TO “ABSTRACT” EXECUTION

RECALL: TEST CASE GENERATION

SYMBOLIC EXECUTION

test suite



THE PROBLEM OF COVERAGE

SYMBOLIC EXECUTION

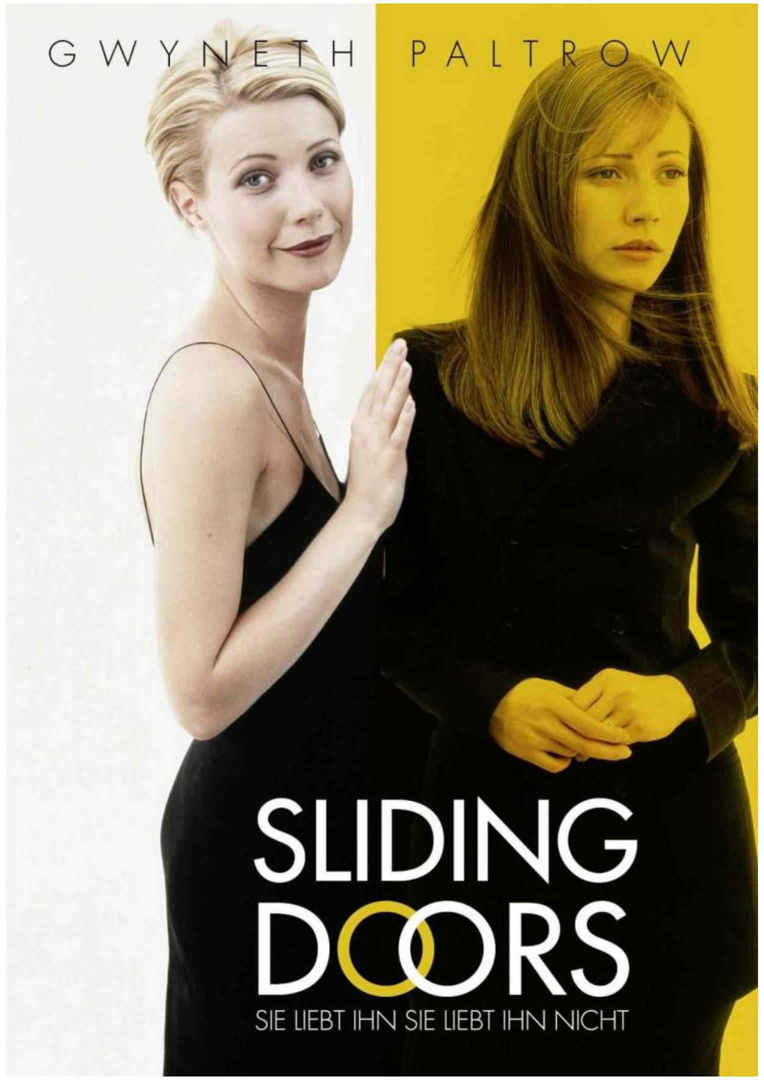
```
#include “stdlib.h”
int main(){
  int c = getchar();
  if (c == 12345){
    blowUpTheOcean();
  } else {
   return 0;
  }
}
```

A divergence, beyond which some behavior will be missed

BIG IDEA: EXPLORE BOTH SIDES

SYMBOLIC EXECUTION

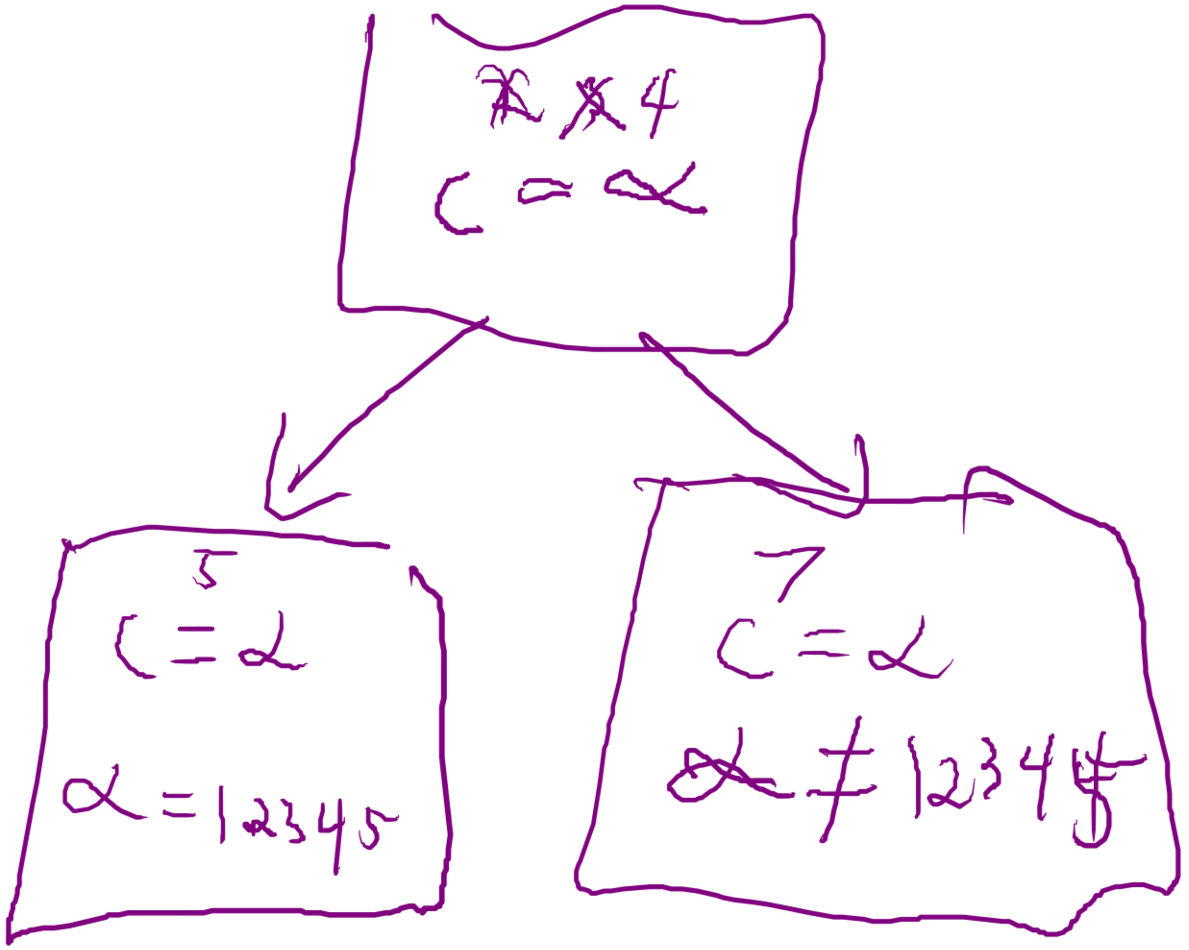
```
#include “stdlib.h”
int main(){
  int c = getchar();
  if (c == 12345){
    blowUpTheOcean();
  } else {
   return 0;
  }
}
```



TECHNIQUE #1: *SYMBOLIC* INPUT

SYMBOLIC EXECUTION

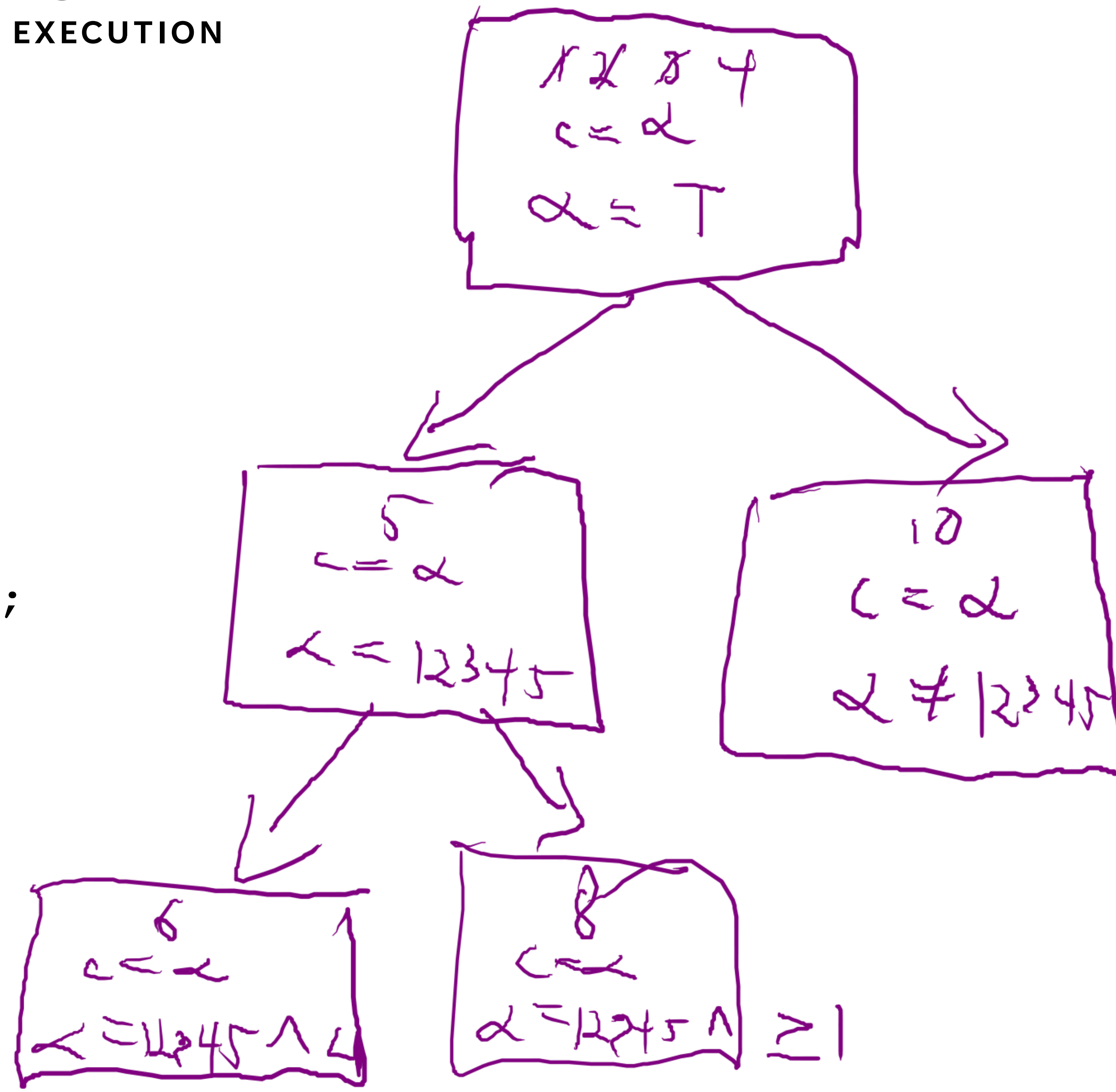
```
1: #include "stdlib.h"
2: int main(){
3:    int c = getchar();
4:    if (c == 12345){
5:       blowUpTheOcean();
6:    } else {
7:     return 0;
8:    }
9: }
```



PATH FEASIBILITY

SYMBOLIC EXECUTION

```
1: #include “stdlib.h”
2: int main(){
3:   int c = getchar();
4:   if (c == 12345){
5:     if (c < 1){
6:       blowUpTheOcean();
7:     }
8 :  } else {
10:     return 0;
11:   }
12: }
```



BIG IDEA #2: PATH CONSTRAINTS

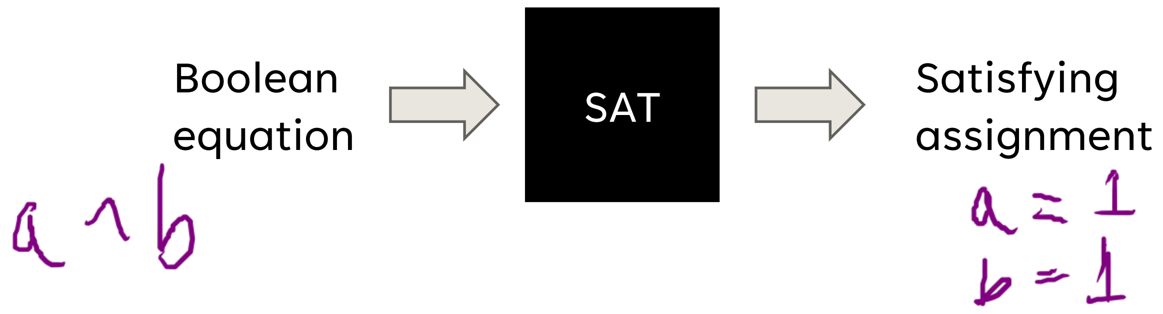
SYMBOLIC EXECUTION

```
#include “stdlib.h”
int main(){
  int c = getchar();
  if (c == 12345){
    if (c < 1){
      blowUpTheOcean();
    }
  } else {
    return 0;
  }
}
```

$\longleftarrow \quad \alpha = 12345 \wedge \alpha < 1$

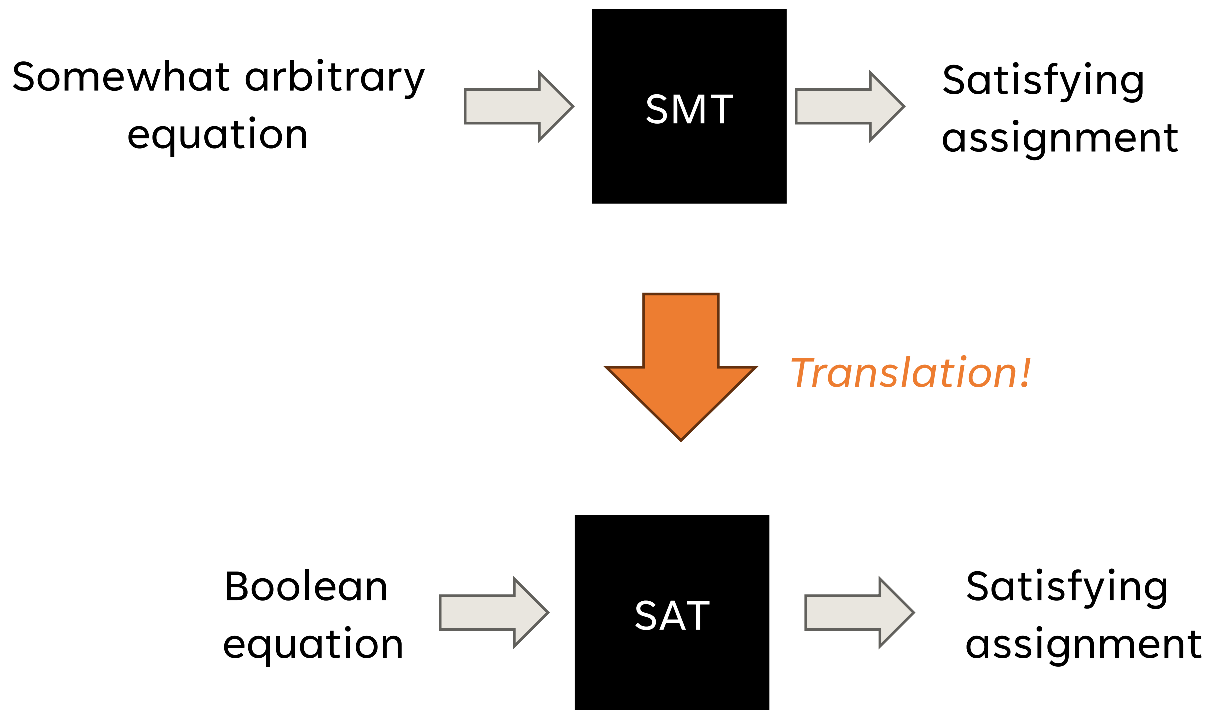
THE MAGIC OF THE SOLVER

SYMBOLIC EXECUTION



THE MAGIC OF THE SOLVER

SYMBOLIC EXECUTION



$a \wedge b \wedge \neg b$

$b \wedge a \vee \neg a$

THE SYMBOLIC EXECUTION TREE

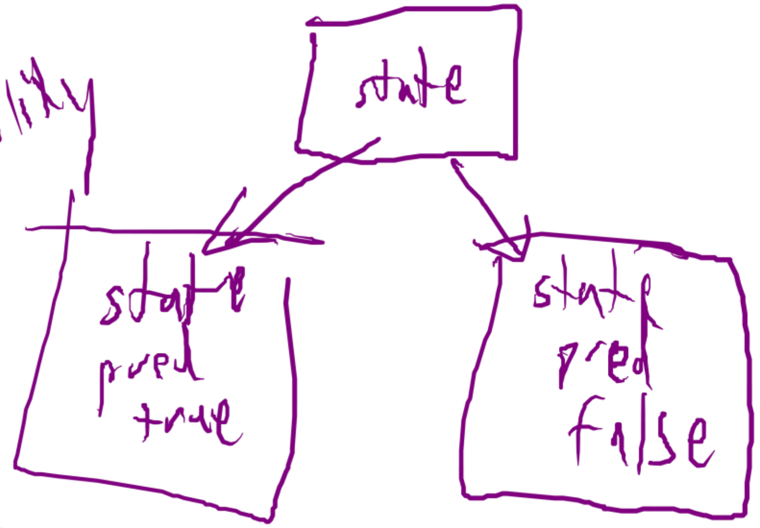
SYMBOLIC EXECUTION

```
#include “stdlib.h”
int main(){
  int c = getchar();
  if (c == 12345){
    if (c < 1){
      blowUpTheOcean();
    }
  } else {
    return 0;
  }
}
```

At each line of the program:

- Advance a symbolic program state
- When you hit a branch split the symbolic state into 2 versions
  - 1) a state that satisfies the predicate
  - 2) a state that does not satisfy the predicate

pick a state, check feasibility



$c = \alpha$

$\boxed{\alpha \geq 1 \wedge \alpha = 12345} \leftarrow$ put into smt

TEST SUITE GENERATION

SYMBOLIC EXECUTION

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

$$\boxed{\begin{array}{l} c = \alpha \\ \alpha = 12345 \end{array}}$$

$d = c + 1$

$c = \alpha$

$\alpha = 12345 \wedge \gamma = \alpha + 1$

$d = \gamma$

$\leftarrow$ $c = \alpha$

$$\boxed{\alpha \neq 12345}$$

SOUNDNESS / COMPLETENESS

SYMBOLIC EXECUTION

```
1. int main(){
2.   int c = getchar();
3.   while ( c ){
4.       if (c == 12345){
5.           blowUpTheOcean();
6.       }
7.   }
8.   return 4;
9. }
```

3.5 c=getchar

Soundness

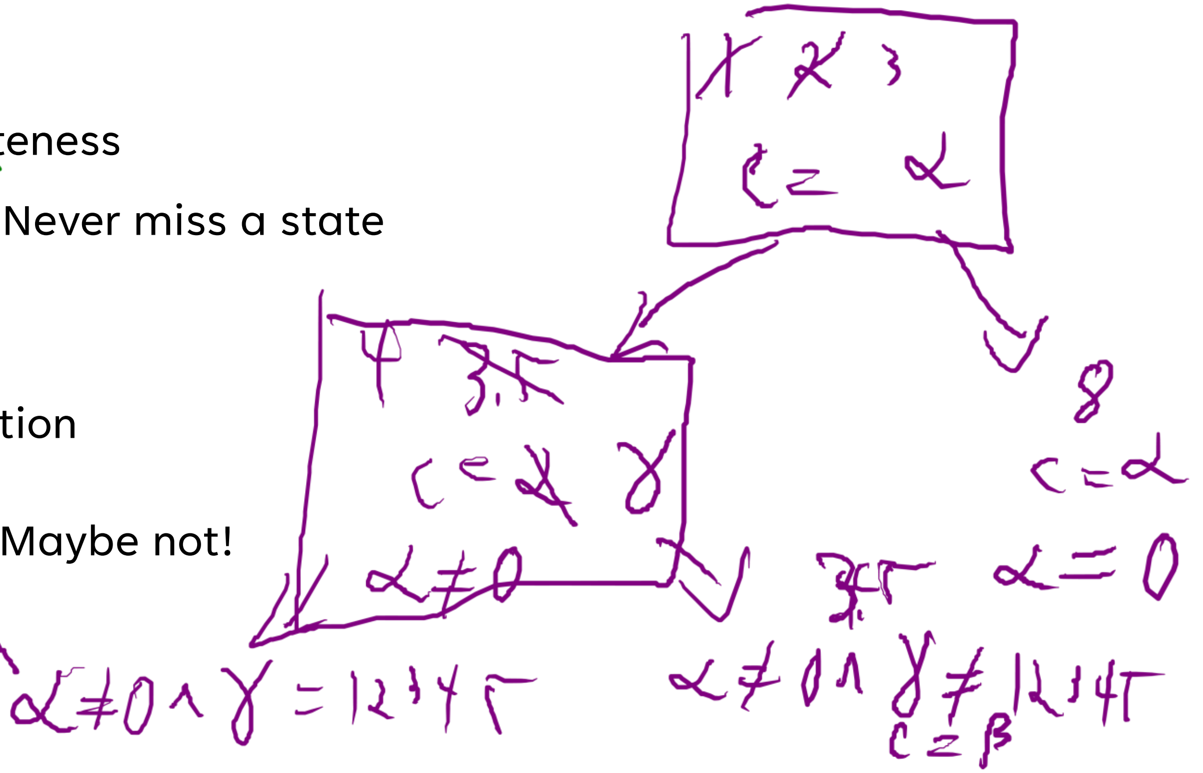
✓ Never generate a state that is unreachable

Completeness

✓ Never miss a state

Termination

bug! Maybe not!



SOUNDNESS / COMPLETENESS

SYMBOLIC EXECUTION

```
int main(){
 int c = getchar();
 while ( c ){
    if (c == 12345){
        blowUpTheOcean();
    }
  }
  return 4;
}
```

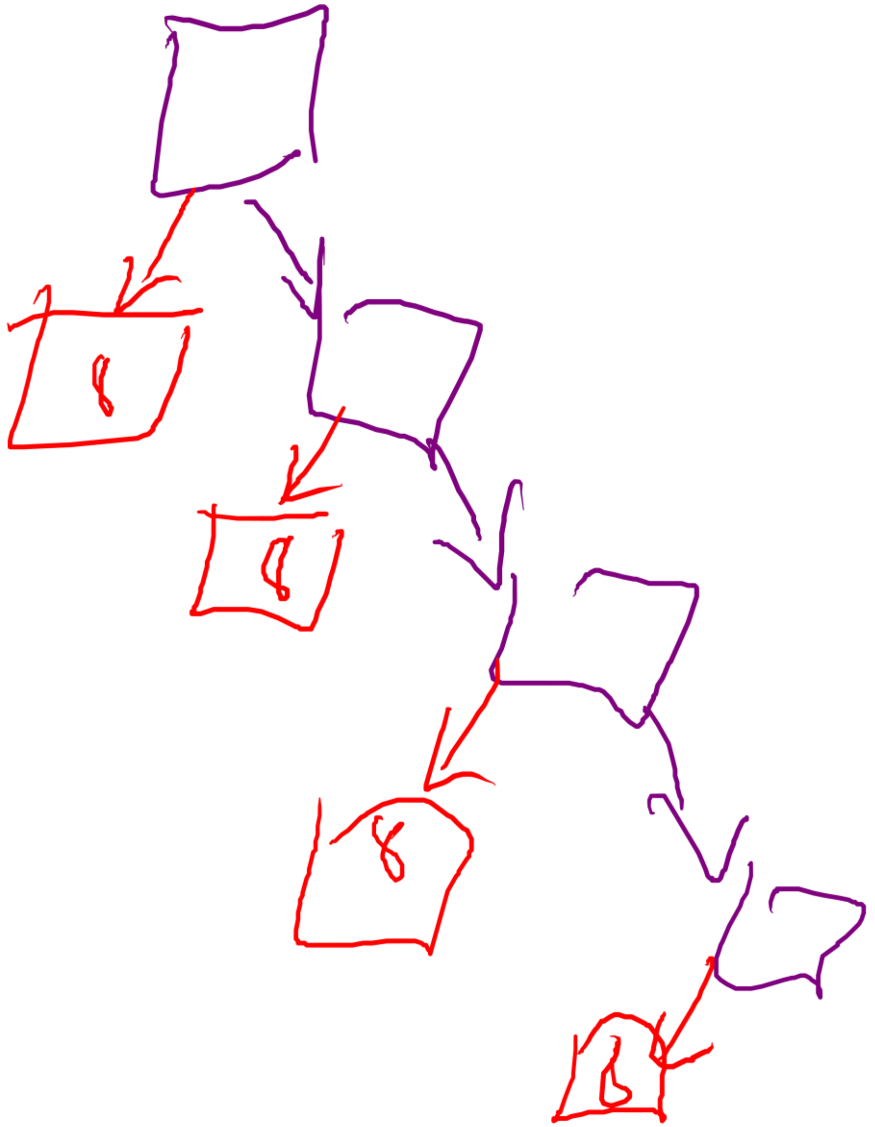
Termination

Maybe not!

LIMITATION: COVERAGE

SYMBOLIC EXECUTION

```
1. int main(){
2.   int c = getchar();
3.   while ( c ){
4.       if (c == 12345){
5.           blowUpTheOcean();
6.       }
7.    }
8.    return 4;
9. }
```



LIMITATION: ENVIRONMENT INPUT

SYMBOLIC EXECUTION

```
int main(){
  int c = getchar();
  c = ioctl(0x92);
  return 4;
}
```

SUMMARY

SYMBOLIC EXECUTION

A simple, elegant idea

NEXT TIME

CONCOLIC EXECUTION

An extension to symbolic execution

Give up on completeness AND soundness to increase practical coverate

coverage