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

↑ can include all details that are not hard-coded into the program and affect the program run.

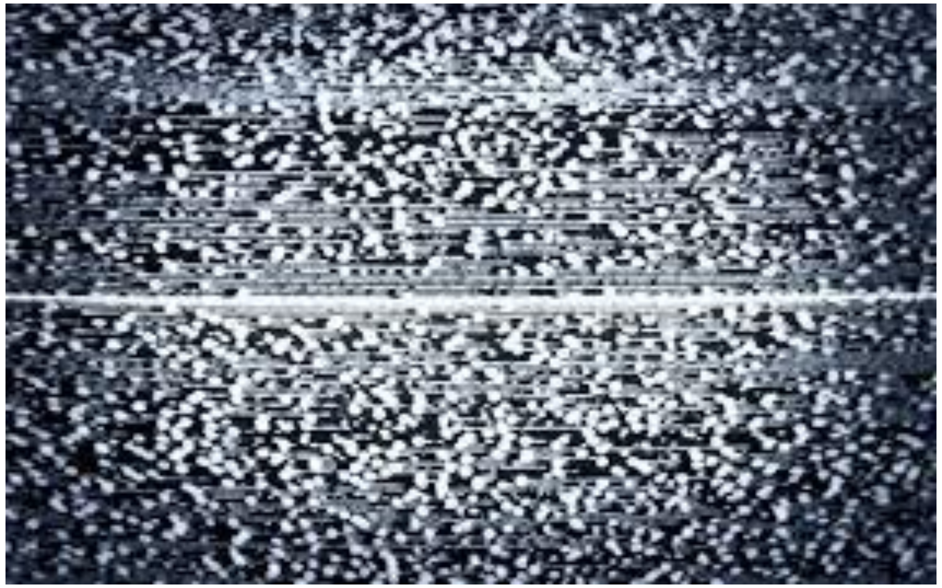
PREVIOUSLY: FUZZING

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?

THIS LECTURE: SYMBOLIC EXECUTION

OUTLINE / OVERVIEW

A METHODICAL APPROACH TO "ABSTRACT" EXECUTION

RECALL: TEST CASE GENERATION

SYMBOLIC EXECUTION

THE PROBLEM OF COVERAGE

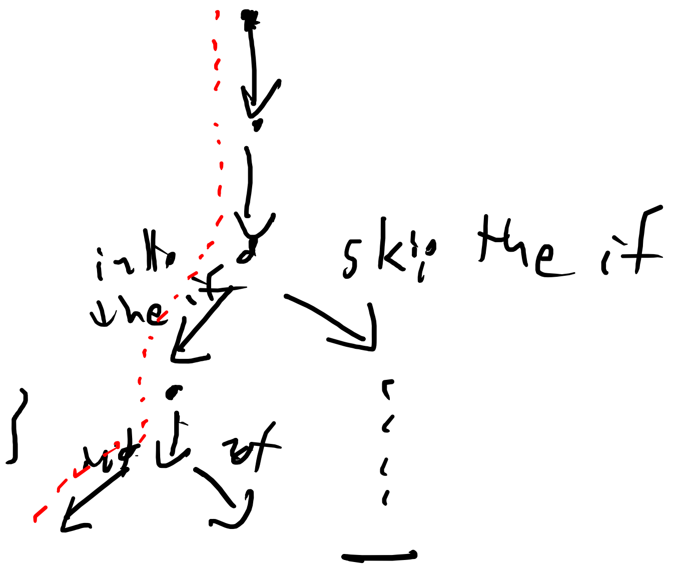
SYMBOLIC EXECUTION

```
#include "stdlib.h"

int main(){
    int c =getchar();
    if (c == 12345){ return 1/0; }
    else {
        return 0;
    }
}
```

PREDICATES GET IN THE WAY!

SYMBOLIC EXECUTION

```
#include "stdlib.h"
 int main() {
      int c = getchar();
      if (c == 12345){
          ~~c=getchar();~~
          if(c==54321){ return 1/0;}
      }

 }
```



ELIMINATING INFEASIBLE PATHS

SYMBOLIC EXECUTION

1) true && true

2) false

```
1  #include "stdlib.h"
2  int main () {
3       int c = getchar();
4    if (c == 12345) {
5         if (c > 54321) {
6              return 0/1;
           }
7       }
```

1 →3

$c = \alpha$ →4

$c = \alpha$

$\alpha == 12345$ →

$c = \alpha$ →7

$\alpha \neq 12345$

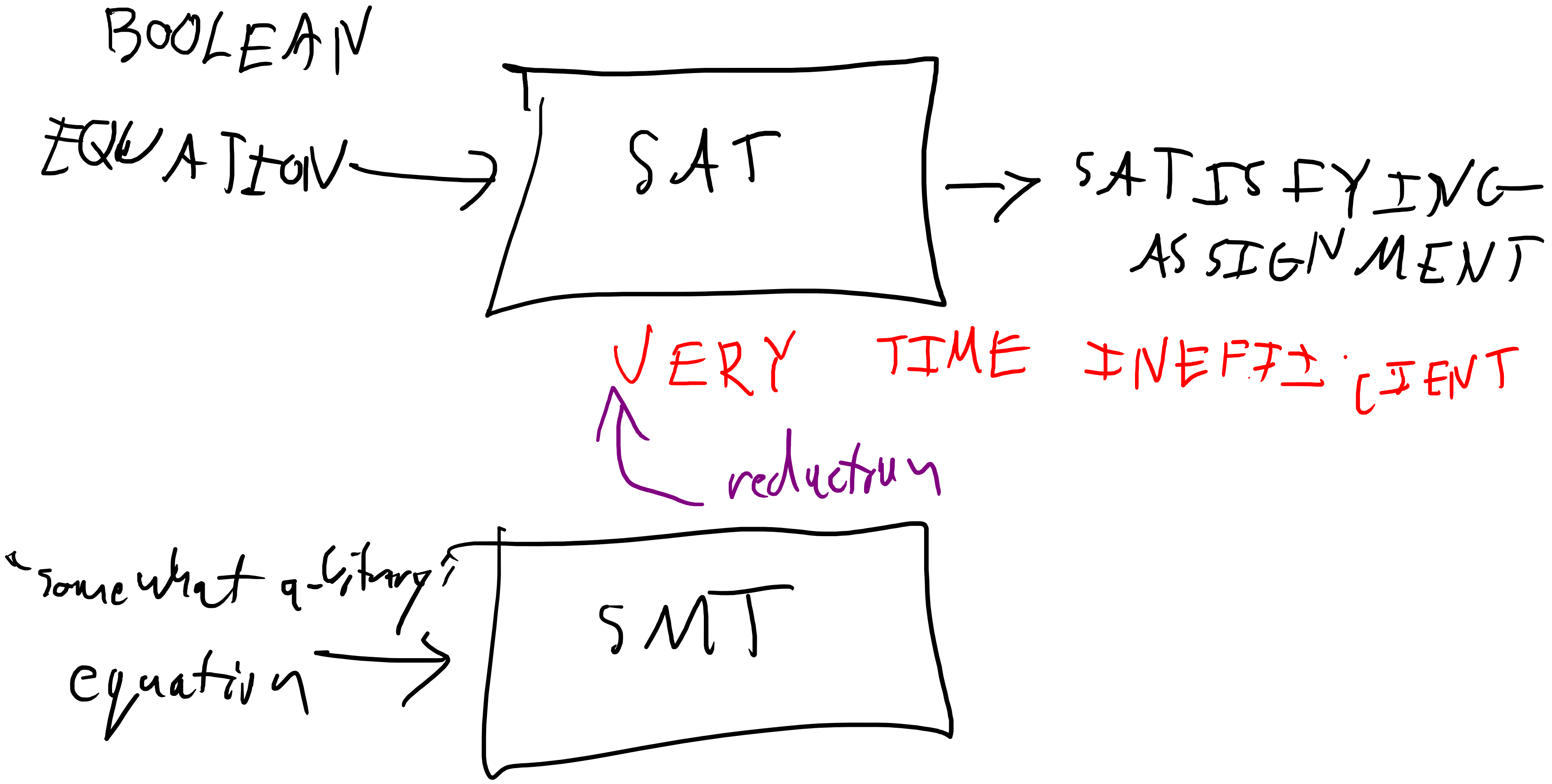
5 3

$c == 12345$ $c \, != 12345$

1) getchar ← 12345 getchar ← α_2

THE MAGIC OF THE SOLVER

SYMBOLIC EXECUTION

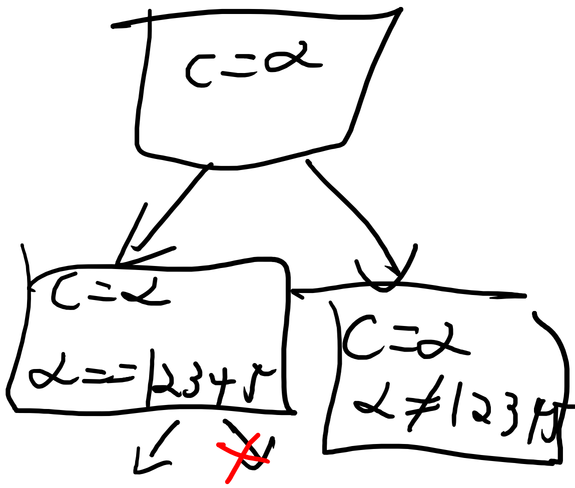


THE SYMBOLIC EXECUTION TREE

SYMBOLIC EXECUTION

At each line of the program:

- advance the symbolic program state
- when you hit a branch, split the symbolic state into 2 versions:
    1) satisfies the branch predicate
    2) does not satisfy the branch predicate



SOUNDNESS / COMPLETENESS

SYMBOLIC EXECUTION

✓ Soundness:

- Never generate a state that violates the constraints

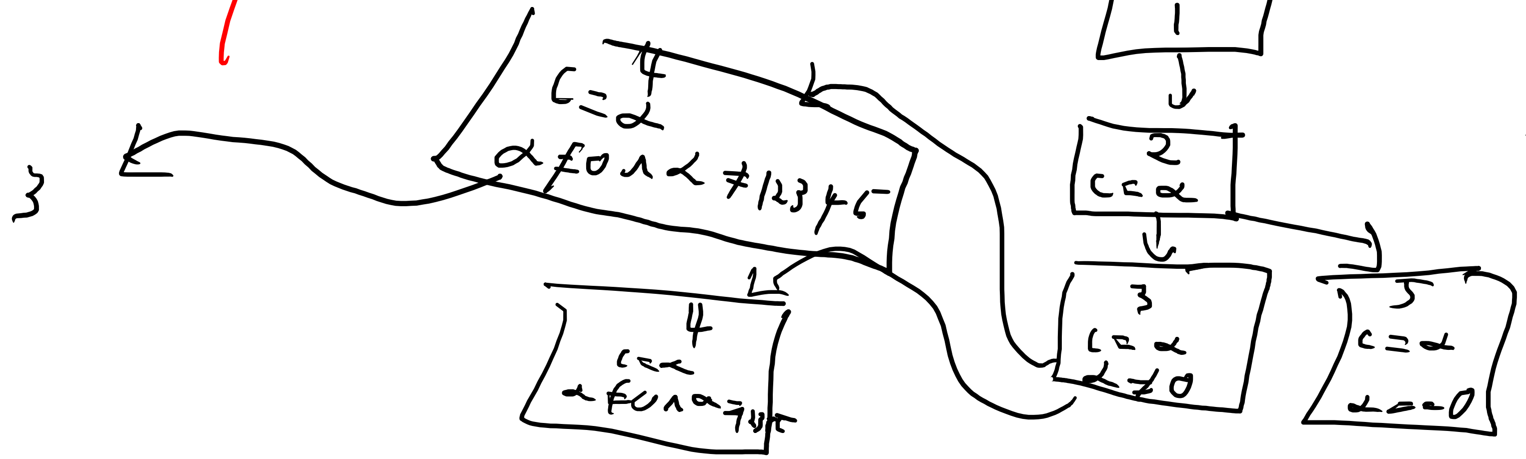
✓ Completeness:

- Never a state we miss

may not terminate

c = getchar

```
1 int main() {
2     int c = getchar();
3     while ( c ){
4         if (c== 12345){B
5 }
```



WRAP-UP

SYMBOLIC EXECUTION

A simple, elegant idea