EXERCISE #30

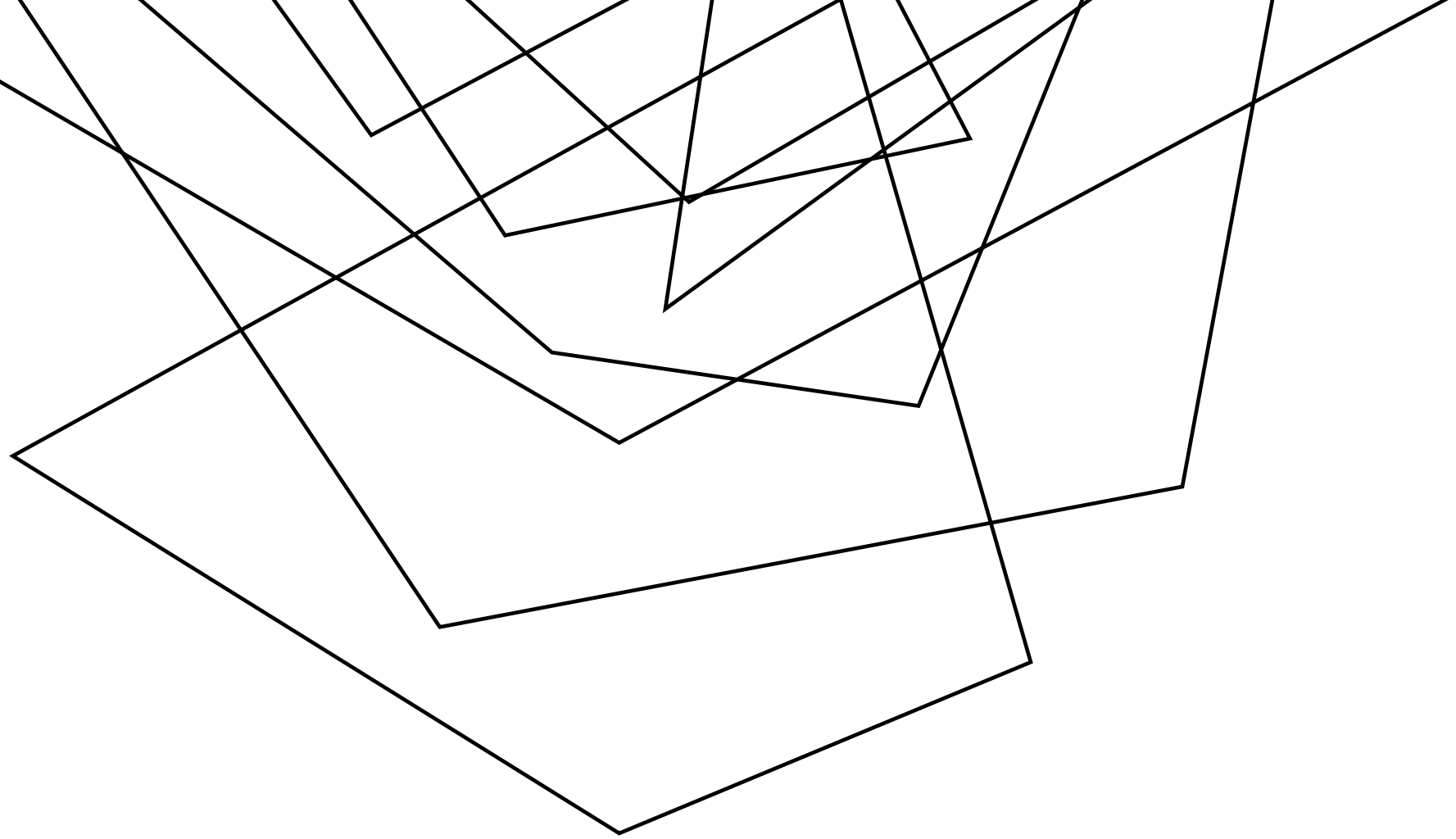
SYMBOLIC EXECUTION REVIEW

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

Draw out the complete symbolic execution tree for the following program. Nodes should be annotated with the path constraint and line number

```
01. int main(){
02.  int c = getchar();
03.  if (c > 2){
04.    if (c < 5){
05.      return 7;
06.    } else {
07.       return 0 / 1;
08.    }
09.   }
10.  return 3;
11. }
```

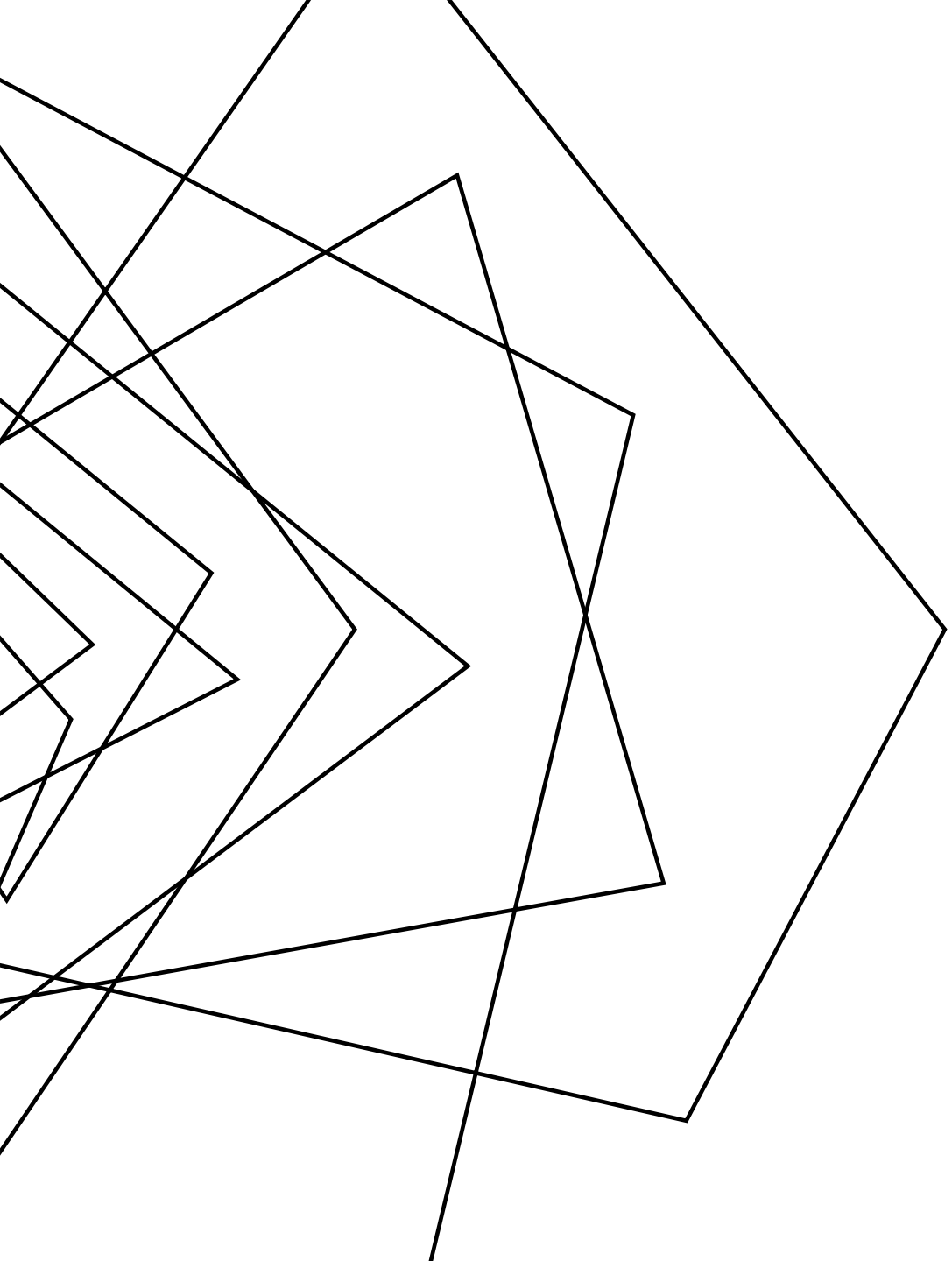
ADMINISTRIVIA AND ANNOUNCEMENTS



CONCOLIC EXECUTION

EECS 677: Software Security Evaluation

Drew Davidson



WHERE WE'RE AT

DYNAMIC ANALYSIS

- generating test cases

PREVIOUSLY: SYMBOLIC EXECUTION

OUTLINE / OVERVIEW

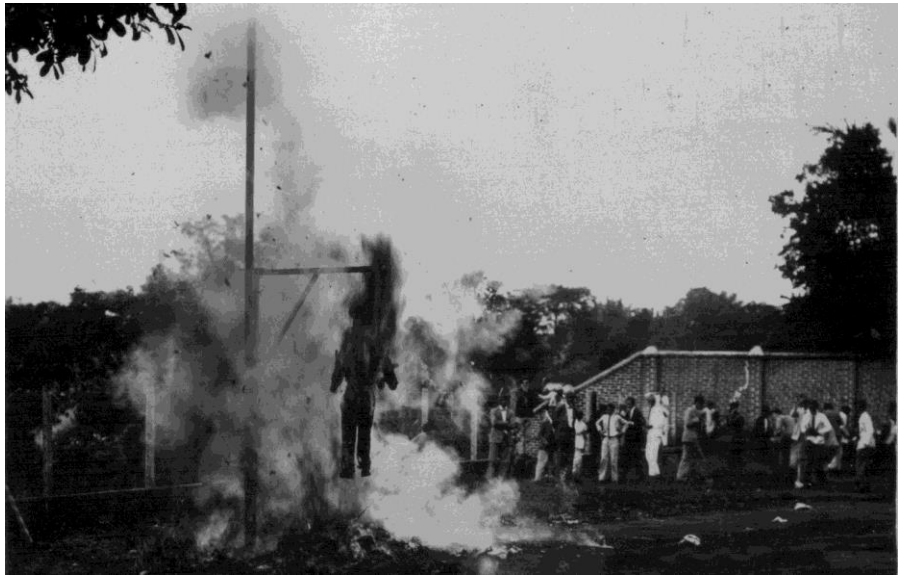
Advance Abstract States across the program

Split abstract states according to predicates to enhance coverage

Use an SMT Solver to determine if the path constraint is feasible

Sound and Complete modulo termination

Stealth caveat of testing as dynamic analysis as well



Symbolic Execution =/= Burning in Effigy

THIS TIME: ENHANCING SYMBOLIC EXECUTION

OUTLINE / OVERVIEW

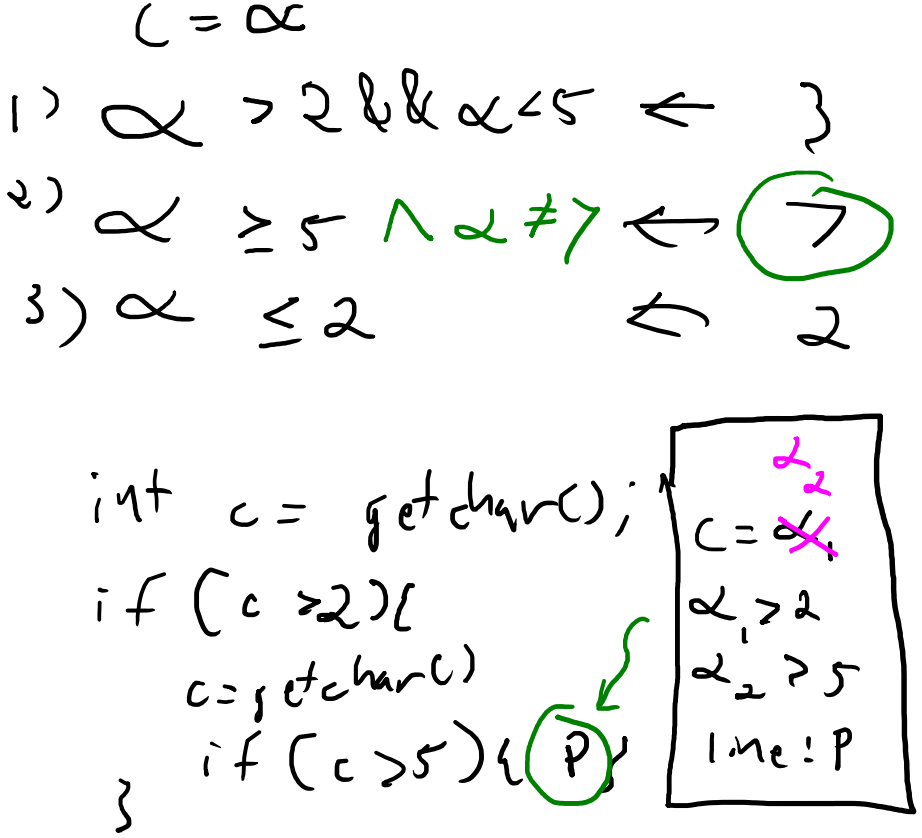
FROM STATE TREES TO TEST CASES

GENERATING TEST CASES

CONCOLIC EXECUTION

WAIT A MINUTE... WE'RE SUPPOSED TO BE BUILDING A TEST SUITE!

... instead, we generated a symbolic execution tree



GENERATING TEST CASES

CONCOLIC EXECUTION

WAIT A MINUTE... WE'RE SUPPOSED TO BE BUILDING A TEST SUITE!

... instead, we generated a symbolic execution tree

```
01. int main(){
02.   int a = getchar();
03.   int b = getchar();
04.   if (a > 5){
05.     return 1;
06.   } else {
07.     return 2;
08.   }
09.   if (b > 3){
10.    return 3;
11.   } else {
12.    return 4;
13.   }
14. }
```

GENERATING TEST CASES

CONCOLIC EXECUTION

WAIT A MINUTE... WE'RE SUPPOSED TO BE BUILDING A TEST SUITE!

... instead, we generated a symbolic execution tree

```
01. int main(){
02.   int a = getchar();
03.   int b = getchar();
04.   if (a > 5){
05.     return 1;
06.   } else {
07.     return 2;
08.   }
09.   if (b > 3){
10.    return 3;
11.   } else {
12.    return 4;
13.   }
14. }
```

FROM TREE TO TESTS

CONCOLIC EXECUTION

EACH STATE'S PATH CONSTRAINT SYMBOLIZES A SET OF TEST CASES

```
01. int main(){
02.   int a = getchar();
03.   int b = getchar();
04.   if (a > 5){
05.     return 1;
06.   } else {
07.     return 2;
08.   }
09.   if (b > 3){
10.    return 3;
11.   } else {
12.    return 4;
13.   }
14. }
```

ASK THE SMT SOLVER FOR A SATISFYING ASSIGNMENT

TERMINATION

OUTLINE / OVERVIEW

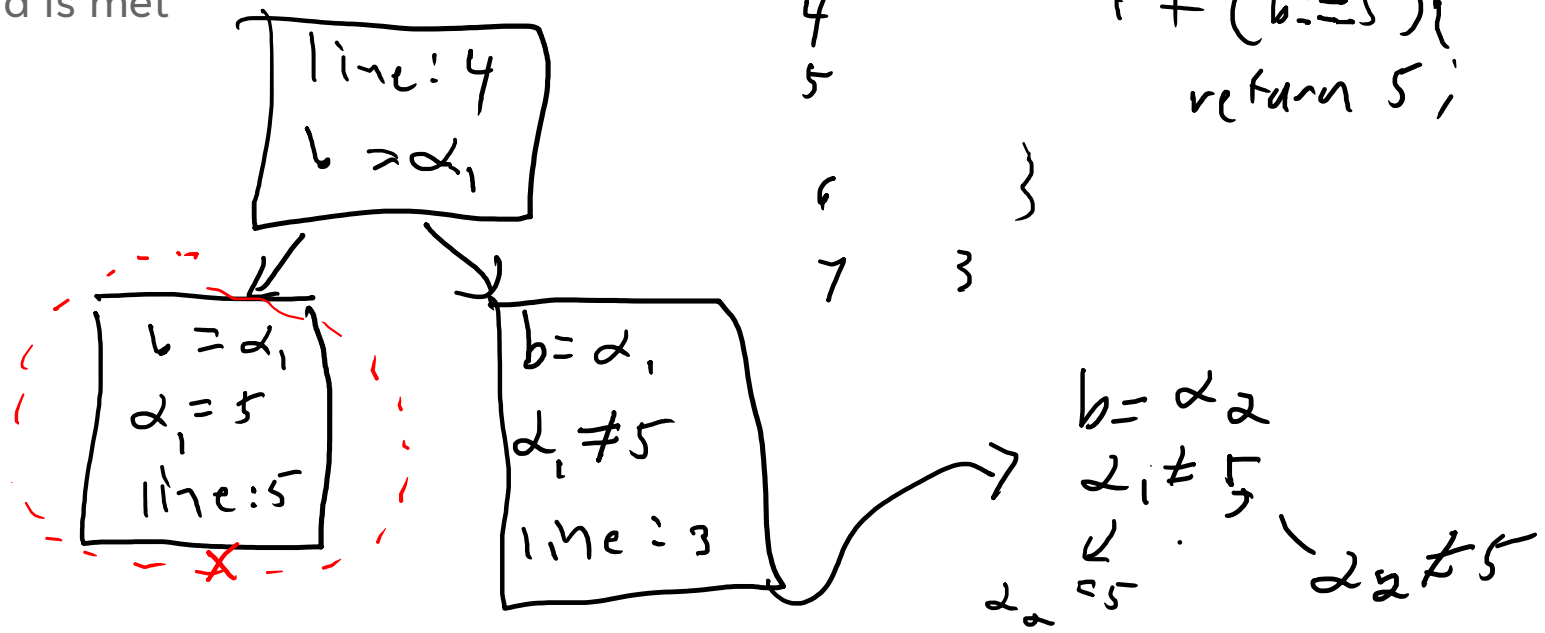
One advantage of symbolic execution:

Partial credit

We can guarantee termination at the expense of completeness

Quit after a certain threshold is met

- Size of the execution tree
- Wall clock time




```
1 int main(){
2     while (true){
3         int b = getchar();
4         if (b==5){
5             return 5;
6         }
7     }
```

STATE PRUNING

SYMBOLIC EXECUTION

Termination Insight: A redundant state has redundant successors

* With proper environmental handling

```
01. int main(){
02.   while (true) {
03.     int b = getchar();
04.     if (b > 5){
05.       return 1;
06.     }
07.   }
08. }
```

RESEARCH DIRECTION: "FIE ON FIRMWARE"

FUZZING

SYMBOLIC EXECUTION FOR "EXOTIC" ENVIRONMENTS

```
int a = *(0x400080);
int b = *(0x400080);
```

STATE PRUNING: LIMITATION

OUTLINE / OVERVIEW

SERIOUS PROGRAMS LIKELY HAVE STATE SPACE EXPLOSION

States are too complicated to prune.

STATE PRUNING: ALTERNATIVES

OUTLINE / OVERVIEW

State Prioritization

Akin to the fuzzing heuristics

STATE PRUNING: ALTERNATIVES

OUTLINE / OVERVIEW

CONCRETIZATION

a = α
b = γ
c = λ
λ > 2 && λ < 7 && α % 2 == 0 && γ[3] > γ[4]

γ[3] = 1
γ[4] = 0

CONCOLIC EXECUTION

OUTLINE / OVERVIEW

Concrete + symbolic

CONCOLIC EXECUTION

OUTLINE / OVERVIEW

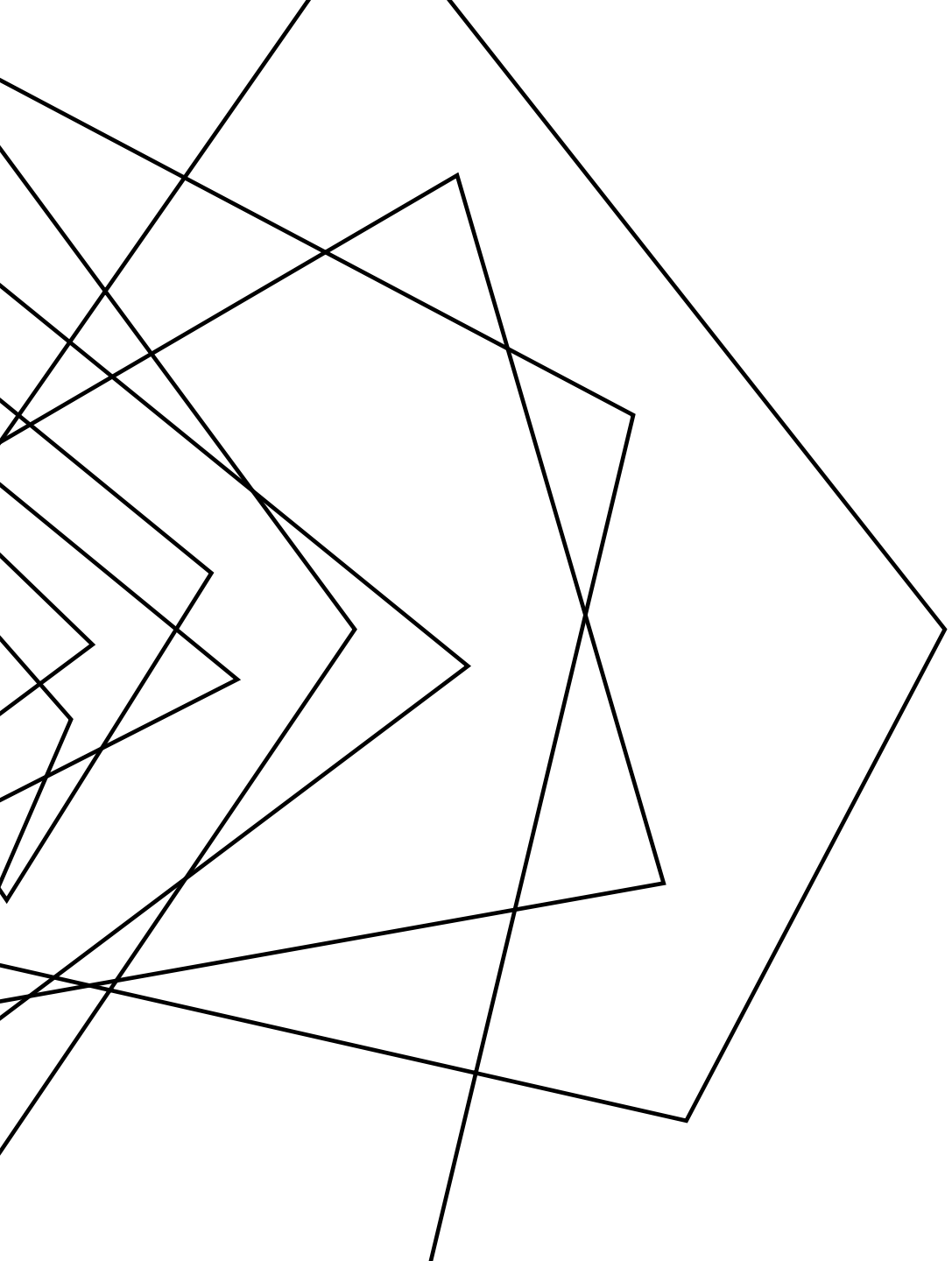
BENEFITS

Increased coverage (at the cost of completeness)

Can still pair with termination thresholds

Much easier to deal with model boundaries

Automatically generating inputs of death



WRAP-UP

SYMBOLIC EXECUTION

A simple, elegant idea

RECALL: TEST CASE GENERATION

SYMBOLIC EXECUTION

THE PROBLEM OF COVERAGE

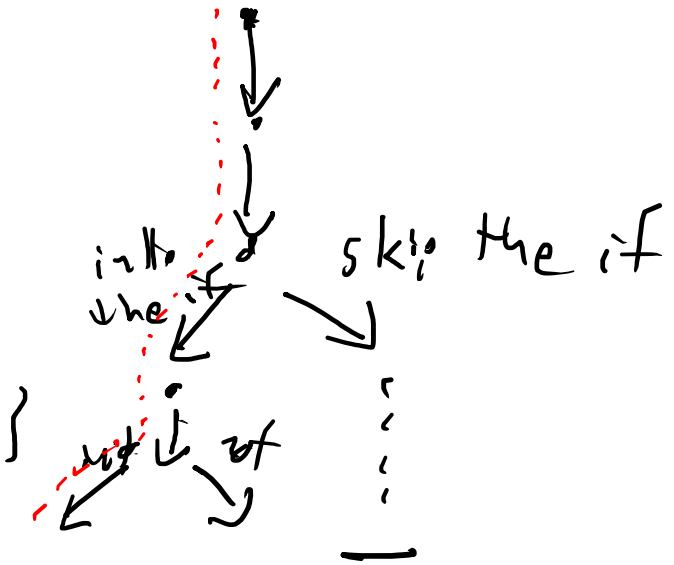
SYMBOLIC EXECUTION

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

PREDICATES GET IN THE WAY!

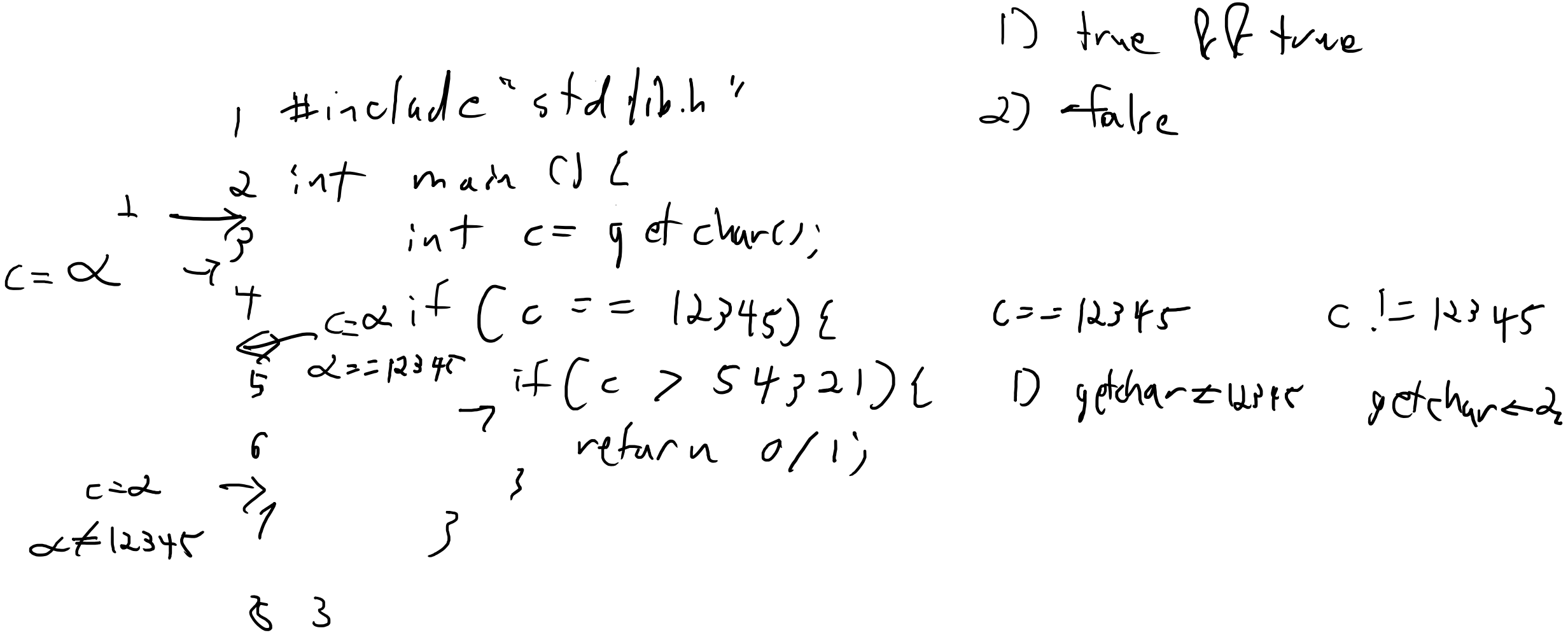
SYMBOLIC EXECUTION

```
#include "stdlib.h"

int main() {
    int c = getchar();
    if (c == 12345){
        c=getchar();
        if(c==54321){ return 1/0;}
    }

}
```



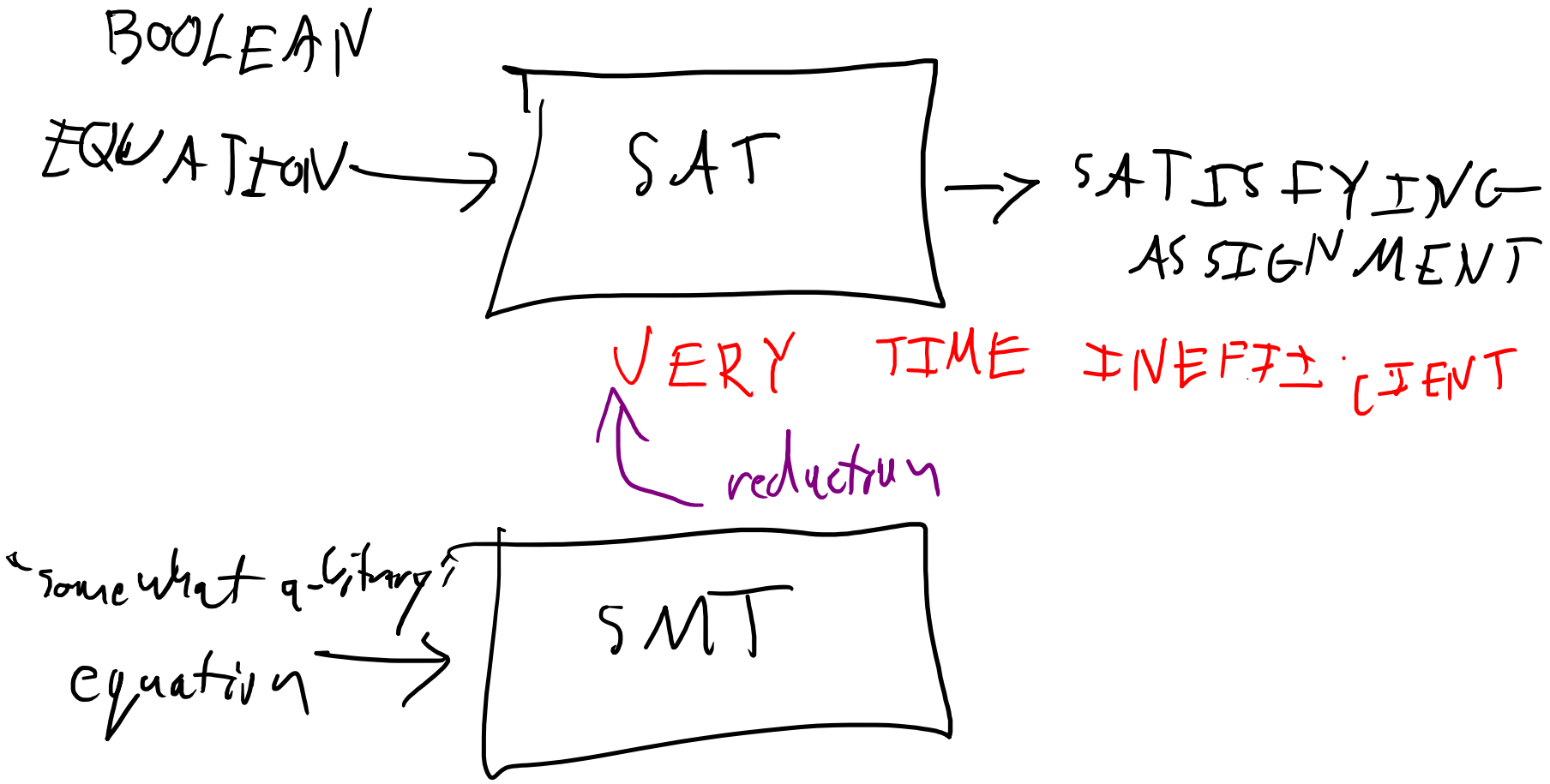
ELIMINATING INFEASIBLE PATHS

SYMBOLIC EXECUTION



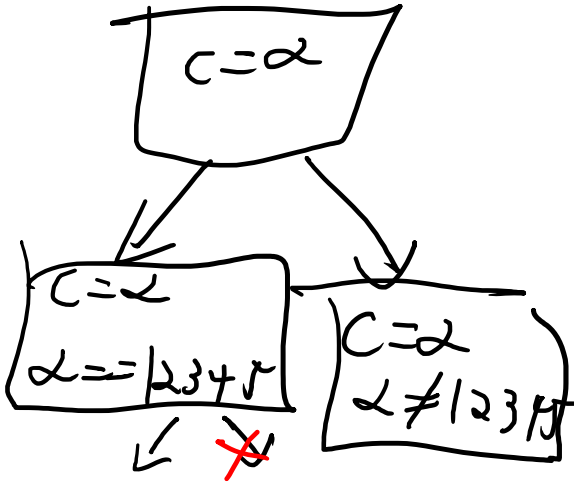
THE MAGIC OF THE SOLVER

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

```
1 int main() {
2     int c = getchar();
3     while (c ) {
4         if (c == 123+5)
5 }
```

c = getchar();