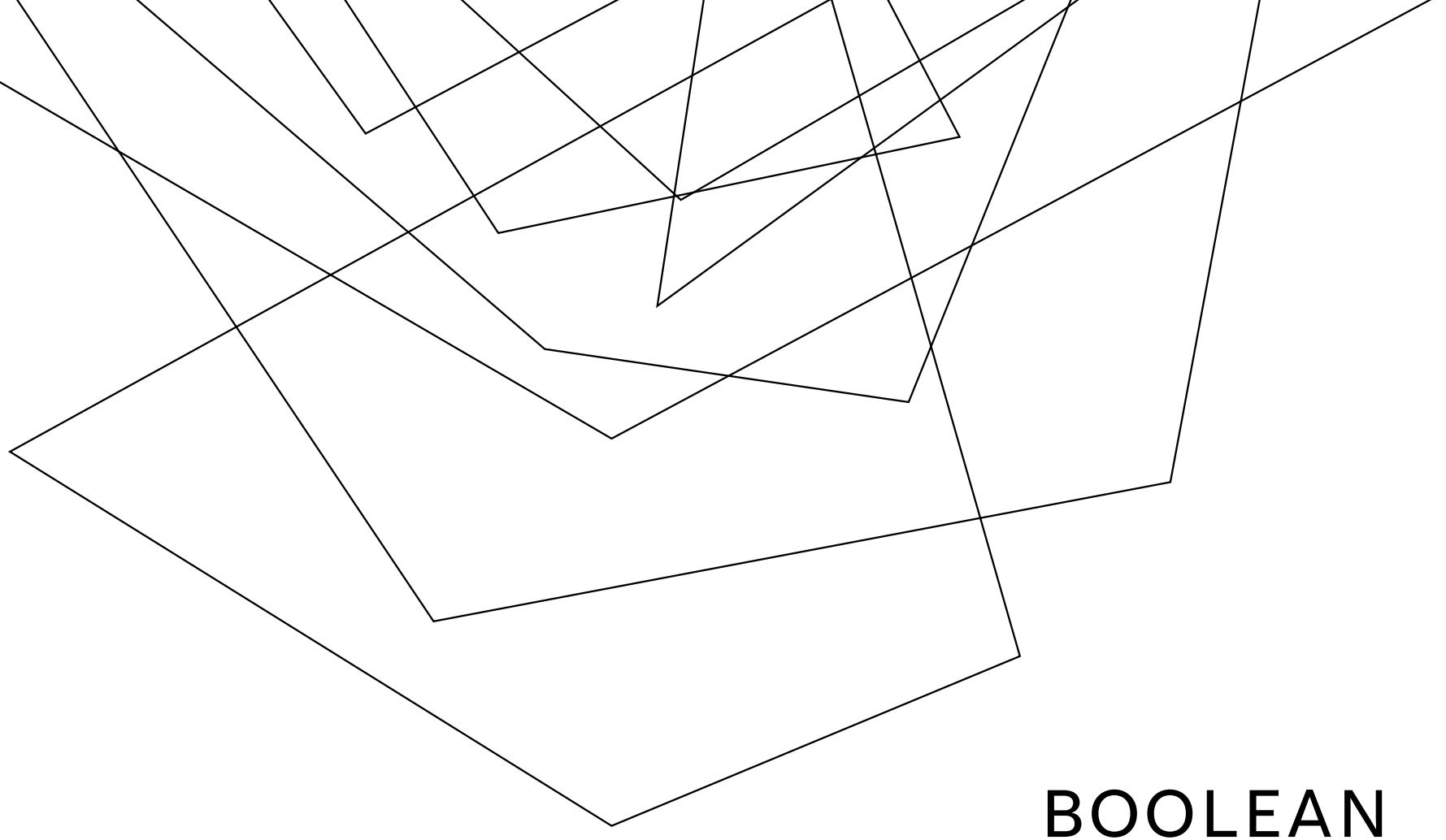
EXERCISE

CONCOLIC EXECUTION REVIEW

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

What is the benefit of concolic execution over symbolic execution? How does it compare in terms of soundness / completeness of vulnerability finding?

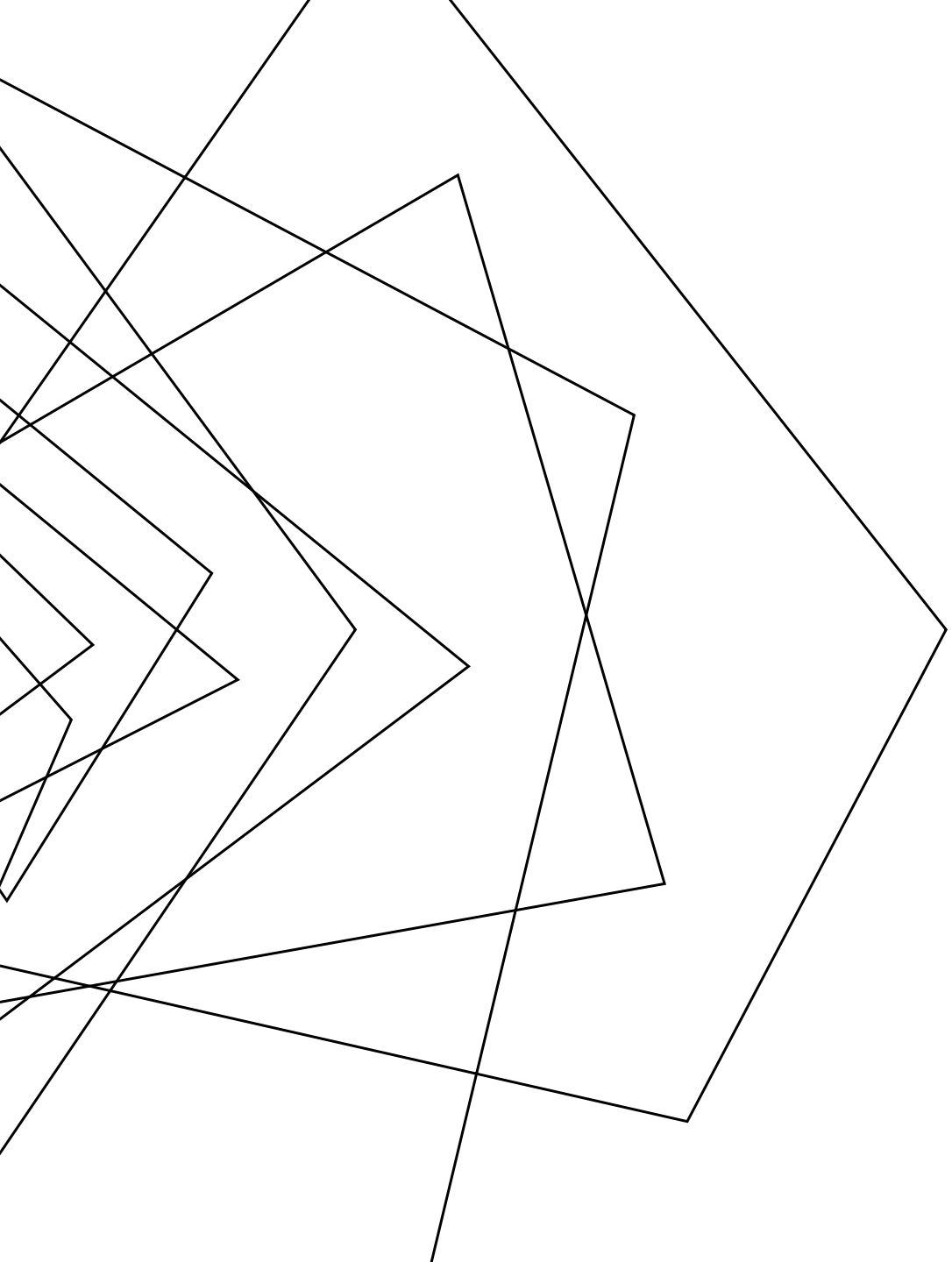
ADMINISTRIVIA AND ANNOUNCEMENTS



BOOLEAN SATISFIABILITY

EECS 677: Software Security Evaluation

Drew Davidson



WHERE WE'RE AT

TOOLS / TECHNIQUES UNDERLYING
SYMBOLIC EXECUTION

PREVIOUSLY: ENHANCING SYMBOLIC EXECUTION

OUTLINE / OVERVIEW

- GENERATING TEST CASES ✓
- PRIORITIZING STATES IN THE SYMBOLIC EXECUTION TREE ✓
- PRUNING DUPLICATE STATES ✓
- CONCRETIZING (SOME) INPUT TO MAKE PROGRESS ✓



THIS TIME: SATISFIABILITY

OUTLINE / OVERVIEW

THE MAGIC THAT MADE SYMBOLIC EXECUTION WORK WAS THE SOLVER

Determines if a path constraint is feasible

Induces a test case that satisfies the path constraint

Allows for consistent concretization



BOOLEAN SATISFIABILITY

SAT AND SMT

AT THE ROOT OF THE SOLVER IS A MECHANISM FOR SOLVING A HARD PROBLEM:

Given a Boolean expression, provide a satisfying assignment to its variables or indicate no such assignment is possible

	A	B	Satisfies
$B \wedge A$	False	False	✗
	False	True	✗
	True	False	✗
	True	True	✓
$B \vee A$	False	False	✗
	False	True	✓
	True	False	✓
	True	True	✓
$\neg A \wedge A$	False	False	✗
	False	True	✗
	True	False	✗
	True	True	✗

Search scales rapidly with the size of the problem

The search for a solution requires a lot of computation

Constant time

Linear time

n log n time

polynomial time

Exponential time

NP
SAT AND SMT

The class of problems where...

A solution can be generated in polynomial time by a nondeterministic Turing machine

A solution can be verified in polynomial time by a deterministic Turing machine

$$P \stackrel{?}{=} NP$$

NP-COMPLETENESS

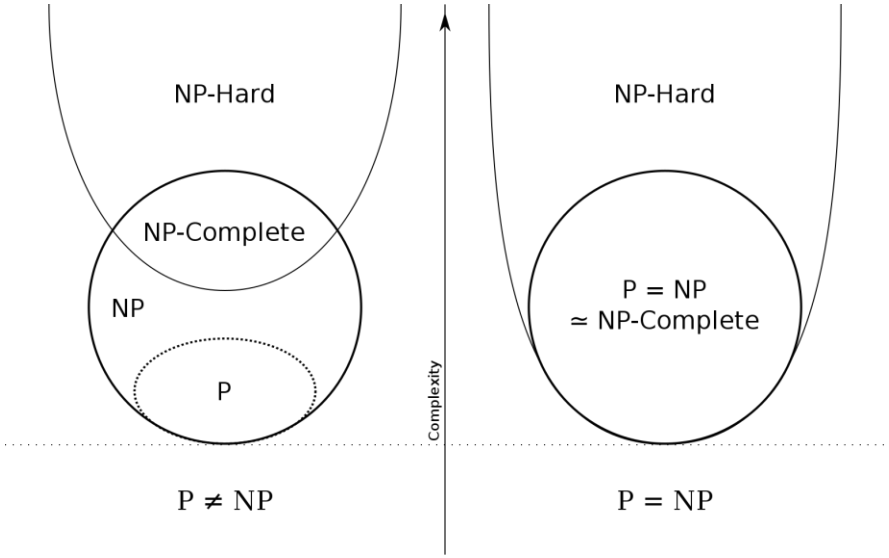
SAT AND SMT

THE CLASS OF PROBLEMS WHERE...

A solution could be used as a solver for any problem in NP

The “most difficult problems in NP”

SAT is the canonical example of an NP-Complete problem



THE MARVEL OF ENGINEERING

SAT AND SMT

NP Reductions once were used to to show that a problem was difficult, now they are used to show that a problem is do-able



HOW DO SAT SOLVERS WORK?

SAT AND SMT

Naïve Solution: Exponential time 2^N

$$(a) \land (b \lor c) \land (\neg a \lor c \lor d) \land (\neg c \lor d) \land (\neg c \lor \neg d \lor \neg a) \land (b \lor d)$$

CONJUNCTIVE NORMAL FORM

SAT AND SMT

✓ $(a \vee b) \wedge (c)$

✓ $(\neg a \vee b) \wedge c$

✗ $(a \wedge \bot) \vee b$

DEFINITION:

A Boolean formula to be in conjunctive normal form (CNF) must be:
A conjunction of one of more *clauses*
where each clause is a disjunction of *literals*
where a literal is a propositional variable or the negation of a propositional variable



CONJUNCTIVE NORMAL FORM

SAT AND SMT

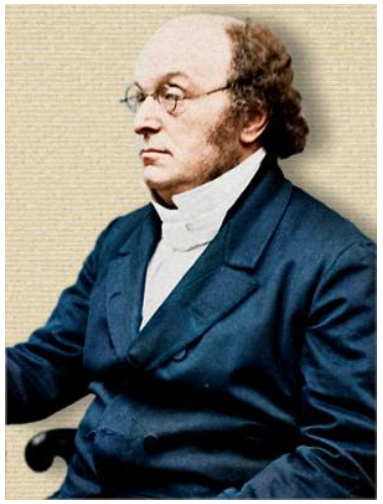
DEFINITION:

A Boolean formula to be in conjunctive normal form (CNF) must be:
A conjunction of one of more *clauses*
where each clause is a disjunction of *literals*
where a literal is a propositional variable or the negation of a propositional variable

ANY BOOLEAN FORMULA CAN BE TRANSFORMED INTO CNF

(Just ask De Morgan)

$\neg(P \vee Q) \Longleftrightarrow \neg P \wedge \neg Q$
$\neg(P \wedge Q) \Longleftrightarrow \neg P \vee \neg Q$
$\neg\neg P \Longleftrightarrow P$
$(P \wedge (Q \vee R)) \Longleftrightarrow ((P \wedge Q) \vee (P \wedge R))$
$(P \vee (Q \wedge R)) \Longleftrightarrow ((P \vee Q) \wedge (P \vee R))$



Augustus De Morgan

CONJUNCTIVE NORMAL FORM

SAT AND SMT

DEFINITION:

A Boolean formula to be in conjunctive normal form (CNF) must be:
A conjunction of one of more *clauses*
where each clause is a disjunction of *literals*
where a literal is a propositional variable or the negation of a propositional variable

THE BENEFIT OF CNF

Every clause must be satisfied
If any single clause cannot be satisfied, neither can the whole formula



SOLVING CNF FORMULAE

SAT AND SMT

LEVERAGING CONSTRAINTS

We'll identify two "features" of a given formula that may simplify solving the formula

Pure literals

Unit clauses



UNIT CLAUSES

SAT AND SMT

A CLAUSE COMPOSED OF JUST ONE LITERAL IS CALLED A **UNIT CLAUSE**

(a) ∧ (b ∨ c) ∧ (¬a ∨ c ∨ d) ∧ (¬b ∨ d) ∧ (¬d ∨ ¬a) ∧ (b ∨ a) ∧ ¬(¬(b))

The satisfiability of the whole formula is threatened by these "tough customers"

STRATEGY: UNIT PROPAGATION

Prioritize satisfying the "tough customer"

Reduce the formula with the assignment that satisfies the unit

Good news! New unit clauses might be created



An absolute unit!

PURE LITERALS

SAT AND SMT

A VARIABLE THAT IS ALWAYS NEGATED OR NEVER NEGATED IN A FORMULA FORMS A **PURE LITERAL**

(a) ∧ (b ∨ c) ∧ (¬a ∨ c ∨ d) ∧ (¬c ∨ d) ∧ (¬c ∨ ¬d ∨ ¬a) ∧ (b ∨ d)

These "easily handled" literals have an obviously-correct choice

STRATEGY: PURE LITERAL ASSIGNMENT

Make the "obvious" choice

For an arbitrary variable X, if the pure literal ¬X is in the formula, assign false to X

For an arbitrary variable X, if the pure literal X is in the formula, assign true to X

PURE LITERALS

SAT AND SMT

Note, it might not be *necessary* (or sufficient) to assign the easy answer to a pure literal, but you'll never need to reverse the decision

(a ∨ b) ∧ (b ∨ c) ∧ (¬b ∨ ¬c) ∧ (b ∨ ¬c) ∧ (¬b ∨ c)

Here, a is pure: it's "easy" to assign true to a
(however, the formula is still unsat)

(¬a ∨ b) ∧ (¬a ∨ c) ∧ (¬b ∨ c) ∧ (b ∨ ¬c)

a = true

b = true

c = true

Here, ¬a is pure: it's "easy" to assign false to a
(however, the formula is satisfiable even if a is assigned true)

DPLL
SAT AND SMT

THE DAVIS-PUTNAM-LOGEMANN-LOVELAND (DPLL) ALGORITHM

Put unit propagation and pure literal assignment together with brute forcing

Basic intuition

Unless you hit a base case...

Apply unit propagation (if applicable)

Apply pure literal assignment (if applicable)

Brute force some remaining literal (if applicable) and recurse



DPLL

SAT AND SMT

$$(a) \land (b \lor c) \land (\neg a \lor c \lor d) \land (\neg c \lor d) \land (\neg c \lor \neg d \lor \neg a) \land (b \lor d)$$

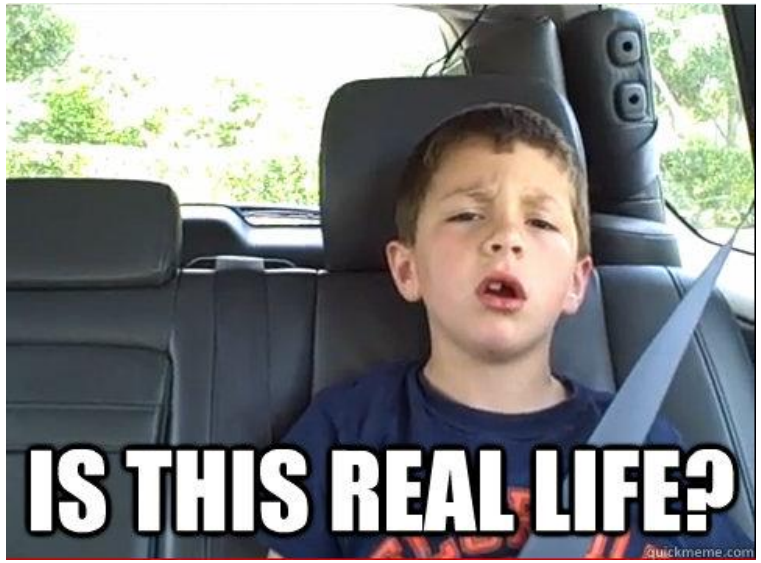
```
function DPLL(φ)
        if φ = true then
                return true
        end if
        if φ contains a false clause then
                return false
        end if
        for all unit clauses l in φ do
                φ ← UNIT-PROPAGATE(l, φ)
        end for
        for all literals l occurring pure in φ do
                φ ← PURE-LITERAL-ASSIGN(l, φ)
        end for
        l ← CHOOSE-LITERAL(φ)
        return DPLL(φ ∧ l) ∨ DPLL(φ ∧ ¬l)
end function
```

REAL-WORLD IMPACT

SAT AND SMT

THE PRACTICAL EFFECT OF DPLL

Symbolic execution for Boolean programs!



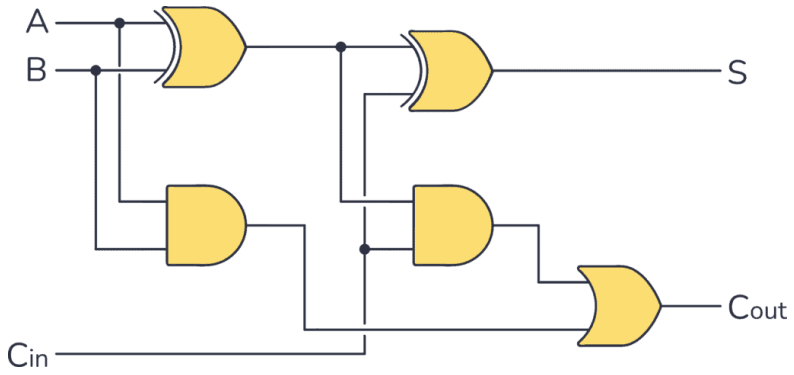
This might seem limited... but on a binary-data computer, every program is a Boolean program

BIT-BLASTING

SAT AND SMT

ENCODE SEMANTICS OF ARITHMETIC OVER FIXED-WIDTH INTEGERS

You already learned how to do this... as freshmen!





These gates have a Boolean representation!

$\neg c_{-1}$

$S_i \leftrightarrow (A_i \oplus B_i) \oplus c_{i-1}$

$c_i \leftrightarrow (A_i \wedge B_i) \vee (c_i - 1 \wedge A_i \oplus A_i)$

NO MAGIC BULLET

OUTLINE / OVERVIEW



WE KNOW SOME CONSTRAINTS ARE COMPUTATIONALLY HARD TO UNPACK

```
int main(){
  char s[80];
  scanf(“%s”, s);
  if (sha256sum(s) == c01b39c7a35ccc3b081a3e83d2c71fa9a767ebfeb45c69f08e17dfe3ef375a7b
}
```

NO MAGIC BULLET

OUTLINE / OVERVIEW

We know some constraints are computationally hard to unpack

```
int main(){
  char s[80];
  scanf("%s", s);
  if (sha256sum(s) == c01b39c7a35ccc3b081a3e83d2c71fa9a767ebfeb45c69f08e17dfe3ef375a7b){
    return 1 / 0;
  }
}
```

FROM SAT TO SMT

OUTLINE / OVERVIEW

Next Time...

Symbolic execution requires path constraints far more complex than Boolean expressions.

Although a naïve reduction is somewhat straightforward, naivety does not gel well with NP-completeness