EXERCISE 30

CONCOLIC EXECUTION REVIEW

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

EXERCISE 30 SOLUTION

CONCOLIC EXECUTION REVIEW

ADMINISTRIVIA AND ANNOUNCEMENTS

Quiz 3 next Friday

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

The search for a solution requires a lot of computation

$B \land A$	B = 1, A = 1	
$B \lor A$	B = 0, A = 1 B = 1, A = 0 B = 1, A = 1	
$\neg A \land A$	No solution	

Constant time

Linear time

n log n time

polynomial time

Exponential time

Search scales rapidly with the size of the problem

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

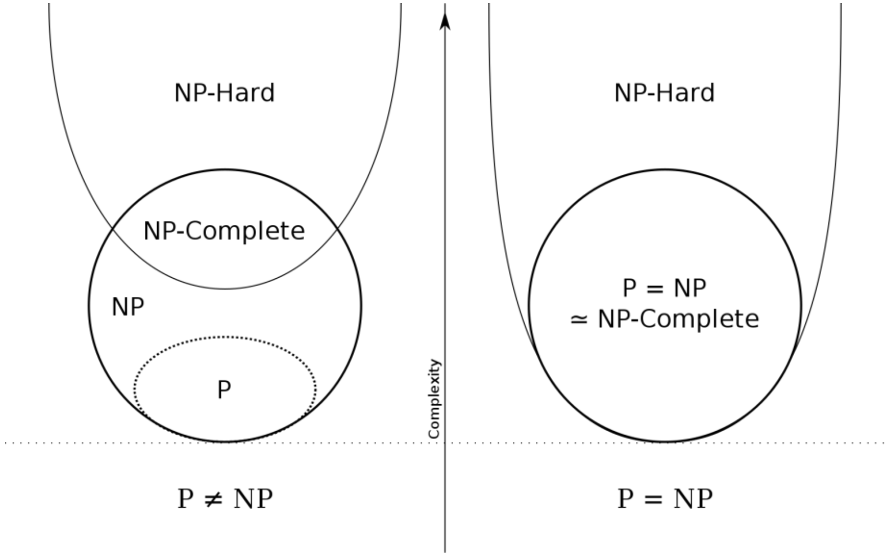
NP-COMPLETENESS
SAT AND SMT

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



A 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

Sriram Rajamani, Microsoft Research

SOLVING SAT

SAT AND SMT

WHAT ARE THE TRUTH VALUES FOR AN ARBITRARY EQUATION?

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

NAÏVE SOLUTION: BRUTE FORCE

Guess every possible assignment of truth values

COMPLEXITY: EXPONENTIAL (2^N)

Intuition: think of the bitvector of length N
where each bit represents a variable (1 for true, 0 for false)

2^N numbers "fit" into N bits

a	b	c	d
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

SOLVING SAT
SAT AND SMT

WHAT ARE THE TRUTH VALUES FOR AN ARBITRARY EQUATION?

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

NAÏVE SOLUTION: BRUTE FORCE

Guess every possible assignment of truth values

COMPLEXITY: EXPONENTIAL (2^N)

Intuition: think of the bitvector of length N
where each bit represents a variable (1 for true, 0 for false)

2^N numbers "fit" into N bits

a	b	c	d
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

CAN WE DO BETTER THAN BRUTE FORCE?

SAT AND SMT

GENERICALLY SPEAKING, NO

THERE ARE COMMON CASES WHERE SHORTCUTS APPLY

STRATEGY: PICK "LOW-HANGING FRUIT"

SAT AND SMT

Some variables **MAY** have "obvious" assignments

Find and assign to easy variables

Reduce the expression

Brute force once no clever strategy remains

CONJUNCTIVE NORMAL FORM

SAT AND SMT

One or more *clauses* joined by **conjunction** where a *clause* is one or more possibly-negated variables joined by **disjunction**

"an AND of ORs"

✓ $(a) \wedge (b \vee c)$

✗ $(a) \wedge \neg(b \vee c)$

✓ $a \vee b \vee c$

✓ $(a) \wedge (\neg b \vee c)$

✗ $(a) \wedge ((b \wedge c) \vee (c))$

✗ $a \vee b \wedge c$

By convention, ***and*** *has higher precedence than* ***or***

✓ $(\neg a) \wedge (\neg b \vee \neg c)$

✗ $\neg(a \wedge b)$

✓ $(a \wedge b)$

HOW CONJUNCTIVE NORMAL FORM

SAT AND SMT

One or more *clauses* joined by **conjunction** where a *clause* is one or more possibly-negated variables joined by **disjunction**

Any Boolean expression can be represented in CNF using the standard Boolean transformations

$\neg(P \lor Q) \Leftrightarrow \neg P \land \neg Q$

$\neg(P \land Q) \Leftrightarrow \neg P \lor \neg Q$

$\neg\neg P \Leftrightarrow P$

$P \lor (Q \land R) \Leftrightarrow (P \lor Q) \land (P \lor R)$

$(P \land Q) \lor (P \land R) \Leftrightarrow P \land (Q \lor R)$

WHY CONJUNCTIVE NORMAL FORM

SAT AND SMT

One or more *clauses* joined by **conjunction** where a *clause* is one or more possibly-negated variables joined by **disjunction**

If one clause is unsatisfiable, the whole equation is unsatisfiable

(**false**) ∧ (…) ∧ (…) ∧ (…) ∧ (…) ∧ (…) ∧ (…) …

This helps to realize our philosophy of “obvious choices” – we’ll try to sort out the most highly constrained clauses

Let’s next consider how to identify and utilitize these cases

THE UNIT CLAUSE

SAT AND SMT

A unit clause or just (“unit”) is a clause with no disjunctions

$(a) \wedge (b \vee c) \wedge (\neg a \vee c \vee d) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d \vee \neg a) \wedge (b \vee d)$

Units are tough customers! There’s only one way to make a unit true, so it yields an obvious choice

This Claus is an absolute unit

LEVERAGING UNITS

SAT AND SMT

Once units have been assigned values, the equation can be reduced

$$(\cancel{a}) \wedge (b \vee \cancel{\neg a}) \wedge (\cancel{\neg a} \vee c \vee b)$$

$a = 1$

$$\cancel{b} \wedge \cancel{(c \vee b)}$$

The new equation might yield more units

$b = 1$

$$(b) \wedge (c \vee b)$$

UNIT PROPAGATION

SAT AND SMT

Unit propagation: continue to identify and eliminate units until

1) There is a satisfying assignment

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

   $\cancel{b} \land \cancel{(c \lor b)}$

2) There is a false clause

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

   $b \land \neg b$

3) There are no more units

   $(\cancel{a}) \land (b \lor c) \land (\cancel{\neg a} \lor c \lor d) \land (\neg c \lor d) \land (\neg c \lor \neg d \lor \cancel{\neg a}) \land (b \lor d)$

UNIT PROPAGATION

SAT AND SMT

Once units have been assigned values, the equation can be reduced

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

The new equation might yield more units

$$(b) \land (c \lor b)$$

Unit propagation: continue to identify and eliminate units until

1) There is a satisfying assignment
2) There is a false clause
3) There are no more units

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

PURE LITERALS

SAT AND SMT

A literal (i.e. variable) that occurs only positively, or only negatively, throughout the entire formula is **pure**

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

Again, a pure literal makes for an obvious choice.

Assigning the “obvious” value to a pure literal doesn’t guarantee satisfiability, but it doesn’t hurt the search for satisfiability

Just like unit propagation, assigning to pure literals may simplify clauses to form new pure literals

PURE LITERAL ELIMINATION

SAT AND SMT

~~Unit propagation~~: continue to identify and simplify out pure literals until

1) There is a satisfying assignment

$$(a \vee \neg b) \wedge (\neg b \vee c)$$

2) There is a false clause

$$(a \vee \neg b) \wedge (\neg b \vee \neg c) \wedge (b \vee c)$$

$$\cancel{\neg b} \wedge \cancel{(\neg b \vee \neg c)} \wedge (b \vee c)$$

3) There are no more pure literals

PUTTING THE STRATEGIES TOGETHER

SAT AND SMT

Unit propagation and pure literal elimination form the core of the most classic sat-solving algorithm, DPLL

DPLL

SAT AND SMT

$\varphi =$ (a) ∧ (b ∨ c) ∧ (¬a ∨ c ∨ d) ∧ (¬c ∨ d) ∧ (¬c ∨ ¬d ∨ ¬a) ∧ (b ∨ 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
```

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
}
```

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