Constraint solving for reverse engineers

Tim Blazytko
⟨tim.blazytko@rub.de⟩

Ruhr-Universität Bochum

23rd February 2017
Today

- What are SMT solvers?
- How do they work?
- What can we do with them?
bool check(uint64_t key) {
    if (key < 7) {
        return (key * 3 > 15);
    }
    return 0;
}

1. $key < 7$

2. $3 \cdot key > 15$

$\Rightarrow (key < 7) \land (key > 5)$

$\Rightarrow key = 6$
**Motivation**

**Complex constraints**

```c
bool check(uint64_t key) {
    return key * key * key * key * key * key * key == 0x90de757572b51cd3;
}
```

We may ask three questions:

- Does a solution exist?
- What is a solution?
- How many solutions do exist?
Motivation
Semantic equivalence

\[ f(x, y) := (x \oplus y) + 2 \cdot (x \land y) \]

We observe

- \( f(1, 1) = 2 \)
- \( f(2, 3) = 5 \)
- \( f(10, 20) = 30 \)

We ask ourselves if

\[ x + y \overset{?}{=} (x \oplus y) + 2 \cdot (x \land y) \]
Satisfiability modulo theories (SMT)

What are SMT solvers?

<table>
<thead>
<tr>
<th>SAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is ((a \lor \neg c) \land (a \lor b \lor c) \land (a \lor \neg b)) satisfiable?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT + modulo theories</td>
</tr>
<tr>
<td>in the best case: NP-complete</td>
</tr>
<tr>
<td>in the worst case: undecidable</td>
</tr>
</tbody>
</table>

Modulo theories

- theory of bit vectors
- theory of arrays

⇒ efficient solvers through conflict-driven clause learning
Conflict-driven clause learning (CDCL)
Algorithm (simplified)

<table>
<thead>
<tr>
<th></th>
<th>Conflict-driven clause learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>choose random assignment</td>
</tr>
<tr>
<td>2</td>
<td>unit propagation</td>
</tr>
<tr>
<td>3</td>
<td>conflict analysis</td>
</tr>
<tr>
<td>4</td>
<td>backtracking</td>
</tr>
</tbody>
</table>

We skip implication graphs and backtracking.
Conflict-driven clause learning (CDCL)

Choose random assignment

\[ g := (a \lor \neg c) \land (a \lor b \lor c) \land (a \lor \neg b) \]

- randomly choose \( a = 0 \)

\[ \Rightarrow (0 \lor \neg c) \land (0 \lor b \lor c) \land (0 \lor \neg b) \]
Conflict-driven clause learning (CDCL)

Unit propagation

\[(0 \lor \neg c) \land (0 \lor b \lor c) \land (0 \lor \neg b)\]

- \(c = 0\)
- \(b = 1\)
- \(b = 0 \downarrow\)

\[\Rightarrow a \lor \neg b \text{ cannot be satisfied}\]

\[\Rightarrow g \text{ cannot be satisfied}\]
Conflict-driven clause learning (CDCL)
Conflict analysis

\[(a = 0, b = 1) \Rightarrow \text{conflict}\]

- \[X \Rightarrow Y \iff \neg Y \Rightarrow \neg X \text{ (contraposition)}\]

\[\Rightarrow \neg \text{conflict} \Rightarrow (a = 1, b = 0)\]

\[\Rightarrow \neg(a \land \neg b) \iff \neg a \lor b\]

\[\Rightarrow cl := \neg a \lor b \text{ (conflict clause)}\]
Conflict-driven clause learning (CDCL)

Next iteration (after backtracking)

\[ g' := g \land cl = (a \lor \neg c) \land (a \lor b \lor c) \land (a \lor \neg b) \land (\neg a \lor b) \]

- randomly choose \( a = 1 \)
- ... 
- randomly choose \( b = 1 \)
- ... 
- randomly choose \( c = 0 \)
- ... 
- SAT
Interaction

\[ g := t_1 \land t_2 \land (t_3 \lor t_4) \]

- \( t_1 : a < b \)
- \( t_2 : a + b == 100 \)
- \( t_3 : b > 50 \)
- \( t_4 : a == 99 \)

- SAT solver randomly sets \( t_4 = 1 \)
- queries theory solver with \((t_1, t_2, t_4)\)
SAT + theory solver

Theory solver

- $t_4 : a = 99$

- $t_2 : a + b = 100 \iff b = 1$

- $t_1 : (a < b) \iff 99 < 1 \not\iff$

- UNSAT

- $\text{cl} := t_1 \lor t_2 \lor t_4$ (conflict clause)
SAT + theory

Final moves

\[
g' := g \land cl = t_1 \land t_2 \land (t_3 \lor t_4) \land (t_1 \lor t_2 \lor t_4)
\]

- SAT solver: \( t_1 = 1, t_2 = 1, t_3 = 1, t_4 = 0 \)
- theory solver
  - \( t_1 : a < b \)
  - \( t_2 : a + b == 100 \)
  - \( t_3 : b > 50 \)

\[\Rightarrow\] SAT for \( a = 1, b = 99 \)

\[\Rightarrow\] SAT
Satisfiability modulo theories (SMT)
What are SMT solvers?

SAT

Is \((a \lor \neg c) \land (a \lor b \lor c) \land (a \lor \neg b)\) satisfiable?

SMT

- SAT + modulo theories
- in the best case: NP-complete
- in the worst case: undecidable

Modulo theories

- theory of bit vectors
- theory of arrays

Slide is duplicated from before
Bit vector

A bit vector \( b \) is a vector of bits with a given length \( l \):

\[
b : \{0, \ldots, l - 1\} \rightarrow \{0, 1\}.
\]

- \( b \mod 2^l, b \in BV \)
- arithmetic operations (\(+, -, *, /, \ldots\))
- bitwise operations (\(\land, \lor, \oplus, \ll, \ldots\))
- \( eax = (eax + ebx) \ll 1 \)
Satisfiability modulo theories (SMT)

Theory of arrays

Operations

- **read**: \( \text{ARRAY} \times \text{INDEX} \rightarrow \text{ELEMENT} \)
- **write**: \( \text{ARRAY} \times \text{INDEX} \times \text{ELEMENT} \rightarrow \text{ARRAY} \)

- \( \text{mov eax, [ebp]} \)
  - \( \text{eax} = \text{read}(M, ebp) \)
- \( \text{mov [ebp], eax} \)
  - \( M' = \text{write}(M, ebp, eax) \)
Applications

Complex constraints

```c
bool check(uint64_t key)
{
    return key * key * key * key * key * key * key == 0x90de757572b51cd3;
}
```

- Does a solution exist? yes
- What is a solution? `0xe80e9aac619831fb`
- How many solutions do exist? 1
Complex constraints

DEMO
Model counting
How many solutions do exist?

Naive approach

1. \( \text{counter} := 0 \)
2. WHILE \( \text{SMT}(\varphi) \in \text{SAT} \):
   1. generate conjunction \( c \) from model assignment
   2. \( \varphi := \varphi \land \neg c \)
   3. \( \text{counter} := \text{counter} + 1 \)

\((k \cdot k \cdot k \cdot k \cdot k \cdot k = 0x90de757572b51cd3) \land (k \neq 0xe80e9aac619831fb)\)

- might not terminate
- does not work for every theory
- independent research branch
Applications

Semantic equivalence

\[ f(x, y) := (x \oplus y) + 2 \cdot (x \land y) \]

We observe

- \( f(1, 1) = 2 \)
- \( f(2, 3) = 5 \)
- \( f(10, 20) = 30 \)

We ask ourselves if

\[ x + y \overset{?}{=} (x \oplus y) + 2 \cdot (x \land y) \]
Semantic equivalence

\[ \varphi = \psi \]

- \( \text{SMT}(\varphi = \psi) \in \text{SAT} \)
  \( \Rightarrow \) single instance that satisfies the constraints
  \( \Rightarrow \) not what we are looking for

- \( \text{SMT}(\varphi \neq \psi) \in \text{UNSAT} \)
  \( \Rightarrow \) no instance that satisfies the constraints
  \( \Rightarrow \) we proved that \( \varphi \) and \( \psi \) are semantically equivalent
Semantic equivalence
Applications
Symbolic execution

\[ \text{add eax, eax} \Rightarrow \text{eax := eax + eax} \]

- perform symbolic computations on basic blocks

\[ \Rightarrow \text{automated derivation of constraints} \]

- query SMT solver to prove characteristics of constraints
Symbolic execution

DEMO
Advanced applications

Graph search

- $t_1 := b \Rightarrow d$
- $t_2 := c \Rightarrow d$
- $t_3 := a \Rightarrow (b \land \neg c) \lor (\neg b \land c)$
- $\varphi := t_1 \land t_2 \land t_3$
Advanced applications

Exploit generation

```c
int vuln(char input[]) {
    char output[15];
    int pass = 0;
    strcpy(output, input);
    if (pass)
        return 1;
    return 0;
}
```

- stack variable pass is set to 0
- vuln returns 1 if pass \(\neq 0\)
- buffer overflow at strcpy overwrites pass
Bounded model checking

Overview

\( \varphi := \text{preconditions} \land \text{prog} \land \neg \text{postconditions} \)

- **preconditions**: initial program state
- **prog**: \( k \) times unwound control-flow graph
- **postconditions**: memory layout for exploitation

- \( \text{SMT}(\varphi) \in UNSAT \): no bug in the bounded program execution
- \( \text{SMT}(\varphi) \in SAT \): bug in the bounded program execution
**Bounded model checking**

**Workflow**

1. create memory dump
2. translate assembly code into intermediate representation
3. inline functions
4. unroll loops
5. apply static single assignment (SSA)
6. apply preconditions and postconditions
7. generate SMT formula
Bounded model checking

DEMO
Advanced applications
Breaking weak cryptography

- *Petya* ransomware
- modified salsa20 cipher
  - 10 instead of 20 rounds
  - operates on 16-bit instead of 32-bit words
- broken by genetic algorithm in 10 to 30 seconds
- SMT solver break it in less than 1 second
Further applications

- deobfuscation
- ROP gadget chaining (compiler)
- shellcode construction
- program synthesis
- ...

Tim Blazytko (RUB)
SMT for reverse engineers
23rd February 2017
General notes

- SMT solvers are very efficient for real-world problems
- different SMT solvers for different use cases
  - boolector for arrays and bit vectors
  - z3 has a powerful API and supports many theories
- generic SMT interface defined by SMT-LIB standard
Limitations

- buggy in some edge cases

  ⇒ try out different SMT solvers

- in general, problems are at least NP-complete

- confusion and diffusion

  ⇒ they cannot break strong cryptography
Conclusion

- SAT solvers
- conflict-driven clause learning
- SAT + theory interaction
- theory of bit vectors and arrays
- solving complex constraints
- model counting
- proving semantic equivalence
- symbolic execution
- graph search
- bounded model checking
- breaking weak cryptography
References


**Tim Blazytko.** *Static data flow analysis and constraint solving to craft inputs for binary programs.*  2015.  URL: https://archive.org/details/static_data_flow_analysis_and_constraint_solving_to_craft_inputs_for_binary_programs.

References


References III

Edward J Schwartz, Thanassis Avgerinos and David Brumley. ‘All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask)’. In: *IEEE Symposium on Security and Privacy 2010*. IEEE. 2010, pp. 317–331.

