Model Checking and Its Applications

Orna Grumberg
Technion, Israel

Research Course, Technion
November 25, 2018
Why Computed-Aided Verification?

It is diverse:

• Logic
• Automata
• Algorithms
• Data structures
• Very efficient implementations that matters
Why Computed-Aided Verification?

Research, development, applications are done in academia, research labs, industry

A large variety of job types

IBM, Intel, Mellanox, Cadence, Microsoft, Apple, Facebook, Amazon, NASA, ...
Model Checking

- Given a system and a specification, does the system satisfy the specification.
Challenges in model checking

Model checking is successfully used, but...

• Scalability
• New types of systems
  - From hardware (finite-state) to software (infinite-state)
• New specifications (e.g. security)
• Applications in new areas
Technologies to help

Developed or adapted by the MC community

- SAT and SMT solvers
- Static analysis
- Abstraction - refinement
- Compositional verification
- Machine learning, automata learning

And many more...
My Current research:

• **Model checking for security:**
  - Finding vulnerabilities of routing protocols, widely used in the Internet
  - A learning-based detector to identify attacks while they happen, at run-time

• **Automated program repair**
  - Simple C programs
  - Given set of mutations
My Current research:

• New compositional methods
  - For hierarchical concurrent C programs with shared variables
  - For communicating programs. Repair if verification failed

• Program difference – modular, deman-driven
  - For evolving programs
  - Did the new version fix a bug? Introduced a new bug?
  - Characterize the difference between versions
How to start

• Take “introduction to software verification”
• Search for a research subject:
  – Interactive paper reading and discussing
• Defining a subject
• More focused paper readings
• Develop your own ideas (together)
Suggested papers to read

• Modular Demand-Driven Analysis of Semantic Difference for Program Versions
  Anna Trostanetski, Orna Grumberg, and Daniel Kroening

• Online Detection of Effectively Callback Free Objects with Applications to Smart Contracts
  Shelly Grossman, Ittai Abramam, Guy Golan-Gueta, Yan Michalevsky, Noam Rinetzky, Mooly Sagiv, Yoni Zohar
Example of a research project
Sound and Complete Mutation-Based Program Repair

[Rothenberg, Grumberg, FM’16]
Mutation-Based Program Repair

- Sequential program
- Assertions in code
- Given set of mutations
- Can we use these mutations to make all assertions hold?
- Assignments, conditionals, loops and function calls
- Assertion violation
- Operator replacement (+ → -), constant manipulation (c → c + 1)
- Return all possible repairs
Example

```c
int f(int x, int y){
    int z;
    if (x + y > 8) {
        z = x + y;
    } else {
        z = 9;
    }
    if (z ≥ 9) z = z - 1;
    assert(z > 8);
    return z;
}
```

$x = 5, y = 2$

$z = 9$

$z = 8$
Example

```c
int f(int x, int y)
{
    int z;
    if (x + y >= 8) {
        z = x + y;
    } else {
        z = 9;
    }
    if (z >= 9) z = z + 1;
    assert(z > 8);
    return z;
}
```

Mutation list:
- Replace + with –
- Replace – with +
- Replace > with ≥
- Replace ≥ with >

Repair list:
- option 1:
  - line 7: replace ≥ with >
- option 2:
  - line 7: replace – with +

Note: Repairs are minimal

At this point z ≥ 9
Example

```c
int f(int x, int y){
1.    int z;
2.    if (x + y > 9) {
3.        z = x + y;
4.    } else {
5.        z = 10;
6.    }
7.    if (z ≥ 9) z = z - 1;
8.    assert(z > 8);
9.    return z;
}
```

Mutation list:
- Replace + with –
- Replace – with +
- Replace > with ≥
- Replace ≥ with >
- Increase constants by 1

At this point $z \geq 10$
Overview of our approach

Finding all correct programs from a finite set of programs

Input: a buggy program

Output: All minimal repairs, sorted by size

\[
\text{int } f(\text{int } x, \text{int } y) \{ \\
1. \quad \text{int } z; \\
2. \quad \text{if } (x + y > 8) \{ \\
3. \quad \quad \quad z = x + y; \\
4. \quad \} \quad \text{else} \{ \\
5. \quad \quad \quad z = 9; \\
6. \quad \} \\
7. \quad \text{if } (z \geq 9) \quad z = z - 1; \\
8. \quad \text{assert}(z > 8); \\
9. \quad \text{return } z; \\
\} \\
\]

Translation → Mutation → Repair

unsatisfiable constraint sets

correct programs

All minimal repairs, sorted by size
First step - Translation

Goal: Translate the program into a set of constraints which is satisfiable iff the program has a bug (i.e. there exists an input for which an assertion fails)

Work by Clarke, Kroening, Lerda (TACAS 2004) (CBMC)
- Simplification
- Unwinding of loops
  - a bounded number of unwinding
- Conversion to SSA
Translation

```c
int f(int x, int y){
    int z;
    if (x + y > 8) {
        z = x + y;
    } else {
        z = 9;
    }
    if (z ≥ 9) {
        z = z - 1;
    }
    assert(z > 8);
    return z;
}
```

\[ \begin{align*}
    g_1 &= x_1 + y_1 > 8 \\
    z_2 &= x_1 + y_1 \\
    z_3 &= 9 \\
    z_4 &= g_1?z_2:z_3 \\
    b_1 &= z_4 ≥ 9 \\
    z_5 &= z_4 - 1 \\
    z_6 &= b_1?z_5:z_4 \\
    z_6 &≤ 8
\end{align*} \]
Second step - Mutation

```c
int f(int x, int y) {
  int z;
  if (x + y > 8) {
    z = x - y;
  } else {
    z = 9;
  }
  if (z ≥ 9) {
    z = z - 1;
  }
  assert(z > 8);
  return z;
}
```

**Mutation list:**
- Replace + with –
- Replace – with +
- Replace > with ≥
- Replace ≥ with >

\[
g_1 = x_1 + y_1 > 8
\]

\[
\begin{align*}
  g_1 &= x_1 + y_1 > 8, \\
  g_2 &= x_1 - y_1 > 8, \\
  g_3 &= x_1 + y_1 ≥ 8, \\
  z_2 &= x_1 - y_1, \\
  z_3 &= 9, \\
  b_1 &= z_4 ≥ 9, \\
  z_4 &= g_1 ? z_2 : z_3, \\
  z_5 &= z_4 - 1, \\
  z_6 &= b_1 ? z_5 : z_4, \\
  z_6 &≤ 8
\end{align*}
\]
Third step - Repair

```c
int f(int x, int y){
    if (x + y > 8) {
        z = x + y;
    } else {
        z = 9;
    }
    assert(z > 8);
    return z;
}
```

\[
\begin{align*}
\{ g_1 &= x_1 + y_1 > 8, g_1 = x_1 - y_1 > 8, g_1 = x_1 + y_1 \geq 8 \\ z_2 &= x_1 + y_1, z_2 = x_1 - y_1 \\ z_3 &= 9 \\ z_4 &= g_1 ? z_2 : z_3 \\ b_1 &= z_4 \geq 9, b_1 = z_4 > 9 \\ z_5 &= z_4 - 1, z_5 = z_4 + 1 \\ z_6 &= b_1 ? z_5 : z_4 \\ z_6 &\leq 8
\end{align*}
\]
**Repair**

\[g_1 = x_1 + y_1 \geq 8\]
\[g_1 = x_1 - y_1 \geq 8\]
\[g_1 = x_1 + y_1 > 8\]
\[g_1 = x_1 - y_1 > 8\]

Choose candidate program of size = 1

\[z_2 = x_1 + y_1\]
\[z_2 = x_1 - y_1\]

Blocking clause for specific assignment

\[b_1 = z_4 \geq 9\]
\[b_1 = z_4 > 9\]

Blocking clause for this assignment

\[z_3 = 9\]

And all other supersets of changes

\[z_6 = b_1? z_5 : z_4\]
\[z_6 \leq 8\]

SAT solver

\[c_1 = 0\]
\[c_2 = 0\]
\[c_3 = 0\]
\[c_4 = 1\]
\[c_5 = 1\]
\[c_6 = 1\]
\[c_7 = 0\]
\[c_8 = 0\]
\[c_9 = 1\]
\[c_{10} = 0\]

UnSAT

\[g_1 = x_1 + y_1 > 8\]
\[z_2 = x_1 + y_1\]
\[z_3 = 9\]
\[b_1 = z_4 \geq 9\]
\[b_1 = z_4 > 9\]

Repair

\[z_5 = z_4 - 1\]
\[z_5 = z_4 + 1\]

SMT solver

\[c_{11} = 0\]
Experimental results on the 41 faulty versions of the TCAS program from the Siemens suite, which implements a traffic collision avoidance system for aircrafts. Comparison to two earlier methods by Konighofer and Bloem implemented in the tool FoREnSiC.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Op. replacement</th>
<th>Level 1</th>
<th>Level 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>{+, −}, {∗, /, %}</td>
<td>{+, −, ∗, /, %}</td>
</tr>
<tr>
<td>Relational</td>
<td>{&gt;, ≥}</td>
<td>{&gt;, ≥, &lt;, ≤}, {==, !=}</td>
</tr>
<tr>
<td>Logical</td>
<td>{</td>
<td></td>
</tr>
<tr>
<td>Bit-wise</td>
<td>{&gt;&gt;, &lt;&lt;}, {&amp;</td>
<td>, ^}</td>
</tr>
</tbody>
</table>

Constant manipulation: C→C+1, C→C−1, C→−C, C→0

Data table showing the comparison of different methods.
Summary

- We suggest a repair method which returns all minimal (bounded) correct programs, in increasing size
  - Based on a given set of mutations

- Minimal mutations: No change is made to the original program unless necessary

- If no repaired program is returned then the given mutations cannot repair the program
Summary

The method can assist a programmer in debugging in initial stages of development

• When bugs are simple, but many

• And also can help beginners
  • Educational tool for students

• Difference analysis can be used to prioritize the returned repaired programs
Questions?