Software Verification

Lecture 01. Introduction to Software Analysis

Ji, Yong-Hyeon

24. 07. 11 (Thu)

Coding & Optimization Together (CO2)
Crypto & Security Engineering Lab (CSE)
Department of Information Security, Cryptology, and Mathematics

Table of Contents

Motivation

1. Motivation

1. Software systems are mostly **unsafe**.

2. Software errors cost the U.S. economy \$60B ($\approx$ 82조) every year.

(1996) Explosion of the Arian-5 rocket. \$8 billion ($\approx$ 11조)

(1998) NASA's Mars climate orbiter lost in space. \$125 million ($\approx$ 1700억)

(2000) Accidents in radiation therapy system. 8 patients died

(2007) Air control system shutdown in LA airport. 6,000 passengers stranded

(2012) Glitch in trading software of Knight Captal. \$440 million ($\approx$ 6000억)

(2014) Airbag malfunction of Nissan vehicles. \$1 million vehicles recalled

1. Motivation

3. Current Technologies for Safe Software

   (a) Code Review

   (b) Testing

   (c) Debugging

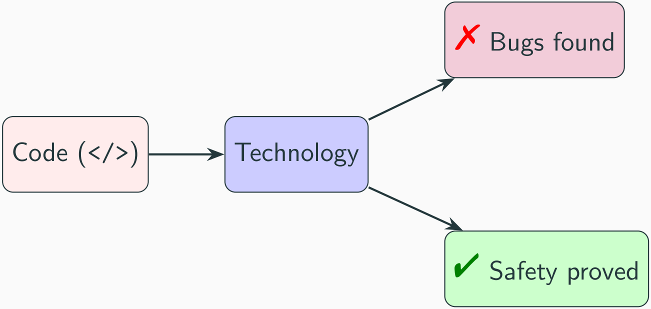
   (d) Simulation

   (e) $\cdots$

Software Analysis

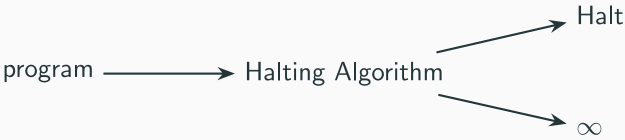
2.1 Software Analysis

- Technology for catching bugs or proving correctness of software.



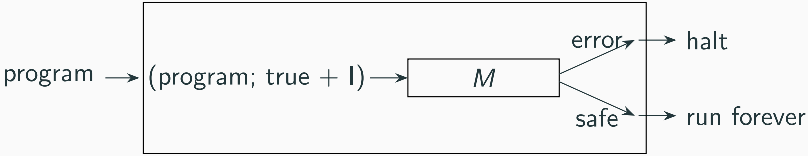
2.2 A Hard Limit

- The Halting problem is not computable. (Alan Turing, 1936)



2.2 A Hard Limit

- If exact analysis is possible, we can solve the Halting problem.

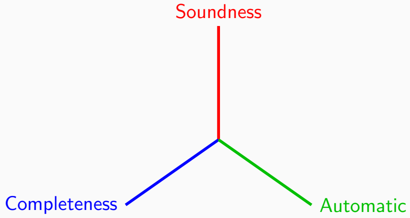


- Rice's Theorem (1951): any non-trivial semantic property of a program is undecidable.

2.3 Trade-off

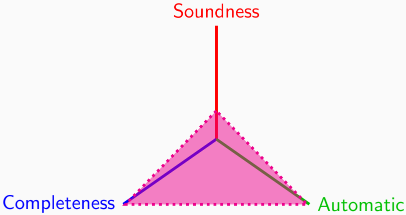
- Three desirable properties
  - **Soundness**: all program behaviors are captured
  - **Completeness**: only program behaviors are captured
  - **Automation**: without human intervention

- Achieving all of them is generally infeasible

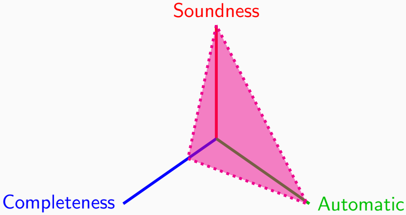


2.3 Trade-off

- Completeness + Automation (Testing)

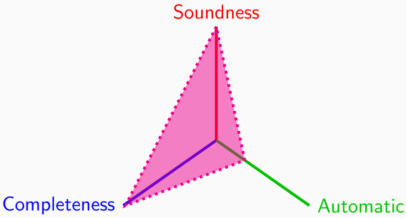


- Sound + Automation (Static Analysis, Compiler)

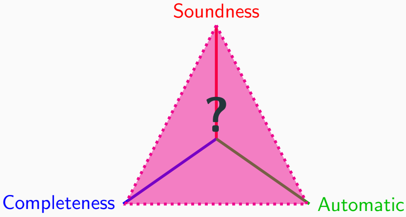


2.3 Trade-off

- Soundness + Completeness (Program Verification)



- And ...



Basic Principle

3. Basic Principle

- Observe the program behavior by “executing” the program
  - Report errors found during the execution
  - When no error is found, report “verified”
- Three types of program execution:
  - **Concrete execution**
  - **Symbolic execution**
  - **Abstract execution**
  - and their combinations, e.g., concolic execution

3. Basic Principle

- **Concrete**

$$2 * 3 - 6 = 0$$

- **Symbolic**

$$a * b + (-c) = 0$$

- **Abstract**

$$(\mathbb{R}, +, *)$$

Software Analysis based on Concrete Execution (Testing)

- Execute the program with concrete inputs, analyzing individual program states separately.

3.1 SW Analysis based on Concrete Execution

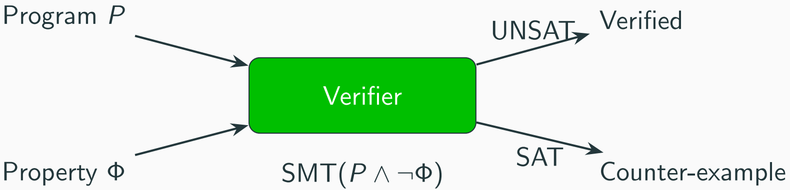
- Error-triggering test? (Test Vector)
- Probability of the error? (assume $0 \le x, y \le 10,000$)
- **Types of Fuzzing**
    - Blackbox Fuzzing
    - Greybox Fuzzing
    - Whitebox Fuzzing

3.2 SW Analysis based on Symbolic Execution

Software Analysis based on Symbolic Execution

- Execute the program with symbolic inputs, analyzing each program path only once.

3.2.1 Example: Symbolic Verification



- Represent program behavior and property as a formula in logic.
- Determine the satisfiability of the formula.

3.3 SW Analysis based on Abstract Execution ($\star$)

Software Analysis based on Abstract Execution (Static Analysis)

- Execute the program with abstract inputs, analyzing all program behaviors simultaneously.

Abstract Interpretation

4.1 Principles of Abstract Interpretation

$$\mathbf{30} \times \mathbf{12} + \mathbf{11} \times \mathbf{9} = ?$$

- Dynamic Analysis (Testing): 459
- Static Analysis: a Variety of Answers
  - "integer", "odd integer", positive integer", "$400 \leq n \leq 500$", etc.

- Static Analysis Process:
  1. Choose abstract value (domain), e.g., $\hat{V} = \{\top, e, o, \bot\}$
  2. Define the program execution in terms of abstract values:

$\hat{\times}$	$\top$	***e***	***o***	$\bot$
$\top$	$\top$	e	$\top$	$\bot$
e	e	e	e	$\bot$
o	$\top$	e	o	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

$\hat{+}$	$\top$	***e***	***o***	$\bot$
$\top$	$\top$	$\top$	$\top$	$\bot$
e	$\top$	e	o	$\bot$
o	$\top$	o	e	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

  3. "Execute" the program:

$$e \mathbin{\hat{\times}} e \mathbin{\hat{+}} o \mathbin{\hat{\times}} o \;=\; o$$

Summary

Summary: Software Analysis

- Basically classified based on how programs are interpreted:
  - Techniques based on concrete execution
  - Techniques based on symbolic execution
  - Techniques based on abstract execution
- Each approach has its own strengths and weaknesses: e.g.,

	Sound	**Complete**	**Automatic**	**When**
Testing/Fuzzing	✗	✔	✔	Dynamic (Runtime)
Symbolic Execution	✗	✔	✔	Static/Dynamic
Static Analysis	✔	✗	✔	Static
Program Verification	✔	✔	✗	Static
?	⋮	⋮	⋮	⋮

Questions?