Software Analysis

- A Journey from Concretization to Abstraction -

Ji, Yong-Hyeon

A document presented for
the Software Analysis

Department of Information Security, Cryptology, and Mathematics
College of Science and Technology
Kookmin University

June 24, 2024

Contents

Chapter 1

Introduction

1.1 intro

Chapter 2

Greybox Fuzzing and Concolic Testing

2.1 Greybox Fuzzing

How to Generate Automatic Test Cases?

2.1.1 Setup

- $\mathcal{P}$: Program under test (PUT)
- Program execution(semantics):
$$[\![\mathcal{P}]\!] : \Sigma^* \to \mathcal{R}$$
  - Σ: input characters, e.g., ASCII Code
  - $\mathcal{R}$: execution results
- Test Oracle:
$$\textsc{Oracle} : \Sigma^* \times \mathcal{R} \to \{\bot, \top\}$$
  - $\top$: the program has run correctly (expected outcome)
  - $\bot$: the program has run incorrectly (unexpected outcome)
  - E.g., "crash oracle", reference implementation, etc.

2.1.2 Random Fuzzing

Algorithm 1: Random Fuzzing

procedure $\textsc{RandomFuzzer}(\mathcal{P})$:
  bugs $\leftarrow \emptyset$
  while time budget expires **do**
    in $\leftarrow \textsc{Sample}(\Sigma^*)$
    res $\leftarrow [\![\mathcal{P}]\!](\text{in})$
    if $\textsc{Oracle}(\text{in}, \text{res}) = \perp$ **then**
      bugs $\leftarrow$ bugs $\cup$ {in}
    end
  end
  return bugs
end

2.1.3 Limitation

- Programs typically expect inputs in specific language ($L \subseteq \Sigma^*$)
  - e.g., web browsers, image processors, compliers, etc.
- Random inputs are unlikely to exercise deep program paths

2.1.4 (Structural) Code Coverage

```
int foo (int x, int y) {
    int z = 0;
    if (x > 3 && y < 6) {
        z = x;
    }
    return z;
}
```

2.2 Concolic Testing

Chapter 3

Propositional Logic

3.1 Propositional Logic

3.1.1 Syntax

- **Atom**: basic elements
  - truth symbols $\bot$("false") and $\top$("true")
  - propositional variables $P, Q, R, \ldots$
- **Literal**: an atom α or its negation $\neg\alpha$.
- **Formula**: a literal or the application of a logical connective (boolean connectives) to formulas

$$
\begin{array}{rcll}
F & \to & \bot & \\
& | & \top & \\
& | & P & \\
& | & \neg F & \text{negation ("not")} \\
& | & F_1 \wedge F_2 & \text{conjunction ("and")} \\
& | & F_1 \vee F_2 & \text{disjunction ("or")} \\
& | & F_1 \to F_2 & \text{implication ("implies")} \\
& | & F_1 \leftrightarrow F_2 & \text{iff ("if and only if")}
\end{array}
$$

- Formula G is a **subformula** of formula F if it occurs syntactically within G.

$$
\begin{aligned}
\mathsf{sub}(\bot) &= \{\bot\} \\
\mathsf{sub}(\top) &= \{\top\} \\
\mathsf{sub}(P) &= \{P\} \\
\mathsf{sub}(\neg F) &= \{\neg F\} \cup \mathsf{sub}(F) \\
\mathsf{sub}(F_1 \wedge F_2) &= \{F_1 \wedge F_2\} \cup \mathsf{sub}(F_1) \cup \mathsf{sub}(F_2) \qquad \vdots
\end{aligned}
$$

- Consider $F : (P \wedge Q) \to (P \vee \neg Q)$. Then

$$\mathsf{sub}(F) = \{F, P \wedge Q, P \vee \neg Q, P, Q, \neg Q\}.$$

- The strict subformulas of a formula are all its subformulas except itself.

- To minimally use parentheses, we define the relative precedence of the logical connectives from highest to lowest as follows:

$$\neg \quad \wedge \quad \vee \quad \to \quad \leftrightarrow$$

- (Currying) Additionally, $\to$ and $\leftrightarrow$ associate to the right, e.g.,

$$P \to Q \to R \iff P \to (Q \to R).$$

- Example:
  - $(P \wedge Q) \to (P \vee \neg Q) \iff P \wedge Q \to P \vee \neg Q$