Software Engineering II - Advanced Topics: Fuzzy Testing

Administrative Details

- Final presentation dates: Nov 30, Dec 5, Dec 7
- Presentation Components:
  - Quick project recap (1-2 minutes)
  - Demo of the project (8-10 minutes)
  - Discussion of the development process (7-8 minutes)
- Evaluation:
  - Student and instructor weighting is 7:3
  - All teams will evaluate each project
  - Link to schedule and more details
  - More information under "Final project presentation" in Assignments

Fuzz Testing (Fuzzing)

- Definition: Automatically generates random inputs (including invalid/abnormal) to discover bad behavior such as crashes without a specific oracle.
- Approach:
  - Construct random/invalid/abnormal inputs
  - Run programs with these inputs
  - Monitor system behavior
  - Identify bugs (e.g., Array indices out of bounds, failing to check null pointers)

Applying Fuzz Testing

- Recent Application: Testing self-driving system using fuzzed images.

Challenges of Fuzz Testing

- Infinite space problem: The set of invalid inputs is unbounded.
- Test case generators must craft cases likely to trigger bugs.

Test Case Generation Strategies

- **Random Generation:**
  - Ineffective due to the inability to penetrate target code, e.g., login functions.
- **Template-based Generation:**
  - Utilizes a grammar or protocol to create more effective test cases.
  - Definitions:
    - Start symbol and expansion rules define how to create inputs.
  - Tools: FuzzingBook Grammars
- **Mutation-based Generation:**
  - Introduces small changes to existing inputs to exercise new behavior.

    - Types of mutation: Insertion, deletion, and replacement.
    - Guided Mutation leverages system feedback to refine mutations and involves an evaluator for efficiency.

Failures Detected by Fuzz Testing

- Types of failures include crashes, endless loops, and resource leaks.

Research on Large Language Model (LLM) for Fuzzy Testing

- Goal: Use an LLM to generate a large amount of test code to find bugs in libraries like TensorFlow.
- LLM's are trained using numerous code snippets, learning syntax and API constraints to generate or mutate valid test programs.

Overview of LLM-based Fuzzy Generation Approach

- Start with seed programs generated by LLMs.
- Perform mutation operations to generate multiple test programs.
    - Masked regions in the code are used for mutation.
    - Types of mutations: argument, prefix/suffix, and method mutation.

Mutation Operator Selection

- Problem: Selecting the best mutation operators to generate valid and unique code snippets.
- Approach: Treat the problem as a Bernoulli bandit problem for effective operator selection.

Fitness Function

- Aims to score generated programs based on depth of execution path, diversity of computation graph, and complexity of API invocations.
- Takes into account:
    - Depth of dataflow graph
    - Number of unique API calls
    - Penalty for repeated API calls with the same inputs

Oracle for Fuzzy Testing

- Executes generated code on different architectures and records variables to detect bugs.
- Types of bugs: Wrong-Computation, Crashes.

Differential Fuzzing

- Technique involving feeding the same input to different applications or implementations to observe execution differences.

Summary - Types of Generators

- Random
- Template-based
- Mutation-based

- Research on fuzzy testing using LLMs

Reference Links for Further Exploration

- Fuzzing Book Resources
- GitLab Coverage Fuzzing
- LLM-based Fuzzy Generation Research Paper

Class

Advanced topic: Fuzzy testing

Instructor: Shaowei Wang

Administrative item

- Final presentation on Nov 30 (Thursday), Dec 5 (Tuesday), and Dec 7 (Thursday)
- See schedule in google spreadsheet
- Each team: ~20 minutes presentation (1-2 min recap, 8-10 min demo, 7-8 min dev process discussion)
- Evaluation: All teams rank projects, 7:3 weight (students: instructor)
- More details Assignments -> Final project presentation

Outline

- Fuzz testing
- Research in fuzz testing

Fuzz Testing (Fuzzing)

- Generates random inputs to discover bad behavior (crashes) without an oracle

Approach

1. Construct random/invalid/abnormal inputs
2. Run programs using these inputs
3. Monitor system behavior and identify bugs
4. Errors found: Array indices out of bounds, null pointer issues, ...

Applying fuzz testing to...

- Recent: Testing self-driving systems
    - Fuzz images

Challenges

- Set of invalid inputs is unbounded
- Fuzzing is an infinite space problem

How to generate test cases?

- Generator-based approaches:
  - Random
  - Template-based
  - Mutation-based

Random

- Ineffective due to test cases being unlike valid input
- Examples: notmecome, wrongcode!124

Template-based

- Creates test cases based on templates with anomalies
- More effective than random test cases

Example Grammar

- Grammar example

Mutation-based

- Introduces small changes to existing inputs
- Mutation: Insertion, deletion, and replacement

Guided Mutation-based

- Leverages system feedback to refine mutations
- Seed, mutation, mutated candidates, target system, selected candidates, evaluator (time/coverage)

Types of Failures

- Crashes
- Endless loops
- Resource leaks or shortages

Oracles for Fuzz Testing

- Fuzzing Book
- GitLab Coverage Fuzzing

Research on Using LLM for Fuzzy Testing

- *Large Language Models are Zero-Shot Fuzzers: Fuzzing Deep-Learning Libraries via Large Language Models* by Zhang et al.
- Goal: Generate test code containing target APIs
- Approach: Use LLM to generate code snippets with APIs for fuzzing DL libraries
  - Read Paper

LLM-based Fuzzy Generation

- Modern LLMs include numerous code snippets from DL libraries in their training
- Implicitly learn Python syntax/semantics and intricate types/constraints of DL APIs
- Generates/mutates valid DL programs for fuzzing

Overview of the Approach

1. Starting from a seed program generated by LLM
2. Perform mutation operations to generate more programs for testing

Mutation Operation Selection

- Identifying mutation operators that help generate more valid and unique code snippets
- Formulated as Bernoulli bandit problem

Fitness Function

- Design fitness function to rank each generated program
- Considers depth of dataflow graph and number of API calls
- FitnessFunction = Depth + UniqueCalls - RepeatedCalls

Oracle

- Execute generated code snippets on CPU and GPU
- Detect potential bugs like Wrong-Computation and Crashes

Differential Fuzzing

- Testing technique that detects bugs by providing the same input to similar applications
- Observes differences in their execution

Coverage

Effectiveness of Operation Selection Algorithm

Bugs Detected

Summary

- Generator-based approaches:
    - Random
    - Template-based
    - Mutation-based
- Research on fuzzy testing