COM S 413x/513x Project 5: Generating and Analyzing Program Invariants

Learning Objectives

1. Understand the concepts of program invariants and specification
2. Understand the challenges of computing program invariants
3. Hands-on experience with a dynamic analysis tool Daikon
4. Inspect and analyze program invariants for real-world applications
5. Connect the knowledge of program changes and versions, bugs and generate test input

Description

Program invariant is one of the most important types of software specification. Diakon is a dynamic analysis tool that can automatically generate such specification. In this project, you will learn how to use Diakon to generate specification for a real-world program. You will also analyze the generated invariants and see how they can help with tasks such as code review, bug understanding and assertion generation. Please use the following steps as guidelines:

1. Get Daikon running
    a. Install the docker that contains Diakon: https://hub.docker.com/r/ashwinkj/daikon
    b. Go to home/DAIKON, find README and example files
    c. Run some test on the examples to make sure Diakon is setup correctly
2. **(15 pt)** Select a case study of your interest (*find.24e2271e* or *grep.db9d6340* see page https://dbgbench.github.io/) and run Diakon to generate invariants:
    a. Retrieve the folders for the corresponding versions of find and grep located in the dbg dockers
    b. Find the test input provided by the dbg benchmark that can trigger the bug (see test/test.sh or you can go to the page https://dbgbench.github.io/ and take a look at “simple bug report”)
    c. Mutate the test input and generate additional 9 similar test inputs (manually) that can trigger the bug
    d. Run the 10 test inputs and generate program invariants at the entry and exit of the buggy function where the fault locations are reported.

e. Patch the buggy programs using one correct fix, one correct but different fix and one incorrect fix (see *Patches* on the dbg web page)
f. Run the same 10 test inputs for the three patched programs and generate invariants for 4 versions of programs

3. **(20 pt)** Analyzing invariants
   a. How many invariants are generated at the entry and exit of a faulty function respectively?
   b. How invariants are changed across different versions? Which invariants are removed, which invariants are added between each pair of two versions?
   c. How many invariants are related to bug or patch?
   d. Please answer the above questions using the following tables

program	Total No. of invariants at the entry and the exit	No. invariants related to bug/patch	Time used to generate invariants
buggy			
correct fix-1			
correct fix-2			
incorrect fix			

Pair of programs	No of invariants added	How many added invariants related to bugs or patches	No. of invariants removed	How many removed invariants related to bugs or patches
buggy-fix1				
buggy-fix2				
buggy-incorrect fix				
fix1-incorrect fix				
fix2-incorrect fix				
fix1-fix2				

4. **(15 pt)** Summarize interesting findings and examples and answer the following questions in a report:

a. (1 pt) Do you find any incorrect invariants reported by Diakon?
b. (3 pt) Compared to code diffs, can invariants comparison better help understand bugs and patches? Why and why not?
c. (1 pt) How long does it take you to inspect the invariants?
d. (3 pt) What are the challenges for you to analyze the invariants
e. (5 pt) Provide a list of added and removed invariants that are related to bugs and patches, and discuss some interesting examples.
f. (2 pt) Any other thoughts and findings? Do you like Diakon? How you can improve Diakon?

Deliverables (50 pt)

Please zip the following files and submit the zipped file to canvas under the "project 5: Generating and Analyzing Program Invariants" column. This project is due **April 26.**

1. **(15 pt)** Generate invariants: from Step 2, you'll submit:
   a. (5 pt) 10 test inputs
   b. (4 pt) Four versions of source code (after applying patches)
   c. (6 pt) All the invariants generated for each version

2. **(20 pt)** Analyze invariants: from Step 3, you'll submit two tables and some explanations of the tables if needed.

3. **(15 pt)** From step 4, you'll submit a report.