Lab 2: Randoop

Summer Semester 2018

Due: 4 June, 8:00 a.m. Eastern Time

Corresponding Lecture: Lesson 4 (Automated Test Generation)

Objective

This lab assignment will familiarize you with Randoop, an automated unit test generation tool that employs feedback-directed **rand**om testing for **o**bject-**o**riented **p**rograms. You will use Randoop to generate error-revealing test cases for a Java program, analyze these test cases to find errors in the source code, fix them, and verify that the errors have been corrected with successive iterations of the tool.

Resources

- Randoop Project Page:
  http://randoop.github.io/randoop/
- Randoop Download Page:
  https://github.com/randoop/randoop/releases/
- Randoop User Manual:
  https://randoop.github.io/randoop/manual/index.html
- Randoop Paper:
  http://homes.cs.washington.edu/~mernst/pubs/feedback-testgen-icse2007.pdf
- Running JUnit tests from the command line:
  http://junit.org/junit4/faq.html#running_4

Setup

The necessary code is located on Canvas in the `randoop.zip` archive. All you need to do to set up this lab is download the archive from Canvas and extract it. Additional libraries required for this lab have been pre-installed on the course VM in the home directory (`/home/cs6340/` or `~/`).

Lab Instructions

In this lab, you will use Randoop to detect and fix all of the bugs in a Java library. The file `randoop.zip` contains the source code for a Java graphing library that contains several errors detectable by Randoop.

It is possible that Randoop may not detect all of the bugs it is capable of finding in a single run (due to programmatic and/or time limitations). Randoop is most effective when run for short periods of time, using multiple iterations of steps 1-6 below:

1. Compile the library source code using the `javac` command in a terminal.
2. Run Randoop on the compiled program to generate error-revealing tests (files named `ErrorTest*.java`).
    - The Randoop user manual contains details for specifying the command-line parameters when running Randoop.
    - Specifying the location of resources needed to run Randoop can be tricky. The easiest way to accomplish this is to run the command to execute Randoop in the directory where you downloaded the library source code with this option:

      ```
      -classpath .:/home/cs6340/randoop-all-3.1.5.jar
      ```
    - For the purposes of this lab you must:
        i. Not generate regression tests.
        ii. Consider unchecked exceptions thrown to be bugs.
        iii. Consider a `NullPointerException` thrown on any input to be a bug.
        iv. Consider a `OutOfMemoryError` or `StackOverflowError` thrown to be expected behavior, and not a bug.

      You will need to adjust the Randoop options to meet these requirements. If you do not set the appropriate flag for each of the above, you may not detect all of the bugs in the library.
3. Compile the newly generated test cases.
    - The test cases generated are designed to be run with JUnit. When compiling and executing the tests, you will need to specify a classpath including the JUnit and Hamcrest jar files in a manner similar to the executing Randoop above.
4. Execute the generated test cases.
    - Refer to running JUnit tests from the command line. Include the directory holding the generated test cases in the classpath.
    - When run, the test cases should all fail. The test cases generated highlight errors in the library. Note that each test may not necessarily point to a distinct error.
5. Use the test cases generated by Randoop to identify and fix the errors discovered by Randoop by modifying the source code of the program.
    - The specification for the library is embedded as comments in the source code. Your bug fixes must be consistent with the provided specification.
    - You must fix the underlying error in the source code causing the test to fail. In particular, you should not suppress the error by removing offending library functions or by modifying the test case generated by Randoop so that it passes.
    - Only fix errors that are discovered by Randoop. If you discover a logical error in the code while inspecting / fixing other bugs, correcting it may throw off the program behavior expected by our grader program.
    - While it may be a valid fix in some contexts, we do not consider the use of try/catch blocks to be the best way to fix any of the bugs in the provided code.

Solutions that use try/catch to either hide or fix the bugs will not be awarded full credit.

6. Verify you have fixed the bugs by recompiling the modified source code and the error-revealing tests. When you execute the test cases as you did in Step 3 above, you should see that all of the tests now pass.

When a new iteration of these steps is unable to produce new error-revealing tests, you may have corrected all of the bugs in the library. You should run a final, long running (5 minutes) iteration to confirm there are no additional bugs to be found.

NOTE: You may or may not see Randoop generate an error-revealing test where a non-static method is invoked on a null object. This is an error within Randoop, not the source code. You may ignore this test case if Randoop generates it. Here is an example of a test case that shows the bug in Randoop, although the variable names and functions called may differ and there may be more code produced in the test case:

```
point76 = null;
point76.Mirror(line90);
```

Items to Submit

1. (100 points) Submit a single compressed file (`.zip` format) named `lab2.zip` containing the full, bug free program source code you generated by iteratively running Randoop and correcting bugs. Submit the following files (even if you did not modify them) directly inside your zip file (no subfolders):
    a. `Line.java`
    b. `Point.java`
    c. `Parabola.java`
    d. `Vector.java`

Similar Tools

- Evo Suite - http://www.evosuite.org/ (Java)