CS 6156 Fall 2020
Homework 0: Introduction to Runtime Verification (RV)

1 Deadline

This homework assignment is due on 9/15/2020 at 11:59pm Anywhere on Earth.

2 Goals and Overview

This homework assignment is a warm-up exercise in which you will set up your development environment and run an RV tool (JavaMOP) on open-source projects. You will

- set up a Docker image for running JavaMOP (you will likely reuse the image throughout the semester);
- learn how to perform RV on open-source projects by running JavaMOP on a toy Maven-based project;
- measure the runtime overhead of JavaMOP on an open-source project (∼150,000 lines of code); and
- **(extra credit)** optionally document your experience with JavaMOP on a project of your choice.

3 Introduction

Gaining experience with RV tools is a learning outcome in this course. So, in this homework, you will create the computational environment needed to run JavaMOP. You will also get your feet wet by running JavaMOP and following the "RV process" that we discussed in Lecture 1.

Parts of this homework are deliberately underspecified and open-ended. The goal is to expose you to the reality of RV research and practice, and to ensure that you are familiar with tools you will need in this class.

4 Tasks

4.1 Setting Up (20 points)

Follow these instructions to set up your environment for running JavaMOP.

1. Install Docker for your platform: `https://docs.docker.com/get-docker`
2. Build a Docker image:

```
$ mkdir hw1 && cd hw1
$ wget https://www.cs.cornell.edu/courses/cs6156/2020fa/resources/mopDockerfile
$ docker build -t hw1:latest - < mopDockerfile # Creates the "hw1:latest" image:
$ docker run -it hw1:latest # start a container in which you will work
```

3. In the Docker container, follow instructions in `$HOME/javamop-agent-bundle/README.txt` to set up a Java agent that attaches JavaMOP to running Java processes. Answer these questions:

(a) How many `*.mop` files are there in the `$HOME/javamop-agent-bundle/props` directory? These are specs that JavaMOP will monitor through the Java agent.

(b) How many of those specs are expressed as ERE, CFG, FSM, LTL, SRS? (Just do a simple case-insensitive directory search, e.g., "`grep -i ere`")

4. Deliverable: Answer 3a and 3b in a file, `setup.txt` and upload to CMS under "Homework 0".

4.2 Apply JavaMOP to a toy Maven-based Java project (20 points)

1. In the Docker container, do the following to run JavaMOP on a toy project:

```
$ wget https://www.cs.cornell.edu/courses/cs6156/2020fa/resources/toy-app.tgz
$ tar xf toy-app.tgz
$ cd toy-app
$ # use Maven to compile the code; capture the command that you used and the Maven output that you got in toy-app/toy-compile.txt
$ # use Maven to run the test(s); capture the command that you used and the Maven output that you got in toy-app/toy-test.txt
$ # in the Maven build file, uncomment the '<plugins>' block that was commented out
$ export RVMLOGGINGLEVEL=UNIQUE
$ # now run the tests again, and JavaMOP will perform RV of the test execution. Capture the Maven output that you get in toy-app/toy-rv.txt
```

2. How do the violations in `toy-app/toy-rv.txt` and `toy-app/violation-counts` differ? Put your answer in `toy-app/violation-difference.txt`

3. Edit `toy-app/src/main/java/edu/cornell/cs6156/App.java` to resolve the violation.

4. Deliverable: submit a single archive file with all the files created during this task.

```
$ mvn clean
$ cd ..
$ tar czvf toy-app-submission.tgz toy-app
$ # upload toy-app-submission.tgz to CMS under "Homework 0"
```

4.3 Apply JavaMOP to an open source project (60 points)

1. Obtain commons-math (`https://github.com/apache/commons-math`), SHA: `450dadfd59`.

2. Follow similar steps as in step 1 in 4.2 to run JavaMOP on commons-math.

3. What is the runtime overhead of JavaMOP?

   (a) run without JavaMOP and measure time, t (Make sure all tests pass.)

   (b) run with JavaMOP and measure time, t_{rv} (Make sure all tests pass.)

   (c) what are the values of t, t_{rv}, and $overhead = t_{rv}/t$?

4. How many unique violations are there? (Number of lines in violation-counts.) NOTE: you should delete violation-counts between runs of JavaMOP or you'll append data from multiple runs.

5. How many times did violations occur? (Sum the first number on lines in violation-counts.)

6. Deliverable: record the answers to items 3c, 4, and 5 in a file, `commons-math.txt` and upload it to CMS under "Homework 0".

4.4 (EXTRA CREDIT) JavaMOP in the wild (30 points)

In the Docker container, apply JavaMOP to your favorite Maven-based Java project from GitHub, and describe your experience by providing answers to the following questions.

1. What is the GitHub URL, and SHA for the project that you chose? Why did you choose this project?
2. Did all tests pass without and with JavaMOP?
   (a) If "yes", what was the runtime overhead?
   (b) If "no", which step(s) failed and how did you (try to) resolve it?
3. How many unique violations are there and how many times did they occur? (These can be non-zero even if tests fail with JavaMOP.)
4. Put answers to 1, 2, and 3 in a file, `in-the-wild.txt` and upload it to CMS under "Homework 0 Extra Credit".