CS 6156 Fall 2020
Homework 2

1 Deadline

This homework assignment is due on 11/17/2020 at 11:59pm Anywhere on Earth.

2 Goals and Overview

This homework is an exercise in writing specifications for runtime verification. You will

- write a specification based on the Javadoc of an API in the standard Java library;
- implement code that satisfies your specification and code that violates your specification;
- check your specification on the code that you implement; and
- **(extra credit)** optionally resolve a potential false alarm in a specification that we saw in class.

3 Introduction

Runtime verification is as good as the specifications that it checks. In class, we discussed the state of the art in obtaining specifications. The goals of this homework are to provide you with experience in writing specifications for runtime verification and to expose you to some challenges involved in doing so. Parts of this homework are deliberately underspecified and open-ended.

4 Tasks

The documentation of `java.util.concurrent.locks.ReentrantLock` [1] says:

> *It is recommended practice to always immediately follow a call to lock with a try block, most typically in a before/after construction such as:*

```
1   class X {
2     private final ReentrantLock lock = new ReentrantLock();
3     // ...
4     public void m() {
5       lock.lock(); // block until condition holds
6       try {
7         // ... method body
8       } finally {
9         lock.unlock()
10      }
11    }
12  }
```

A specification engineer has reasoned about this API documentation and come up with a property called `Lock_Unlock`: "methods must release all locks that they acquire".

In the Docker container from HW-0, fetch the code and specifications required for this homework:

```
$ mkdir hw2 && cd hw2
$ wget https://www.cs.cornell.edu/courses/cs6156/2020fa/resources/hw-2-bundle.tgz
$ tar xf hw-2-bundle.tgz
```

4.1 Implement code that satisfies/violates the `Lock_Unlock` property (15 points)

Complete the code skeleton in the `hw-2-bundle/toy-app` directory.

1. Make the code compile.
2. In `edu.cornell.cs6156.App#failedLockedCount`, add code that violates the `Lock_Unlock` property (i.e., resolve `TODO 1`).
3. In `edu.cornell.cs6156.AppTest#testFailedLockedCount`, implement a passing test that invokes `edu.cornell.cs6156.App#failedLockedCount` (i.e., resolve `TODO 2`).

4.2 Write a JavaMOP specification of the `Lock_Unlock` property (60 points)

Complete the JavaMOP specification skeleton in the `hw-2-bundle/two/Lock_Unlock.mop` file. Note that in the specification engineer's interpretation of the `Lock_Unlock` property, you are not required to check that calls to `unlock()` happen inside `finally` blocks. You may use any of the specification languages that we discussed in class. But, your `Lock_Unlock`.mop specification must

1. generate events and cause monitor synthesis when monitored against the provided `edu.cornell.cs6156.AppTest#testLockedCount`;
2. NOT be violated when monitored against the provided `edu.cornell.cs6156.AppTest#testLockedCount`;
3. be violated when monitored against your implemented `edu.cornell.cs6156.AppTest#testFailedLockedCount`; and
4. NOT miss violations when monitored against your classmates' and Owolabi's implementation of `edu.cornell.cs6156.AppTest#testFailedLockedCount`.

4.3 Monitor your `Lock_Unlock`.mop against your code from 4.1 (15 points)

Follow the same steps (and use the same script) as in HW-0 to build a JavaMOP agent from the two JavaMOP specifications in `hw-2-bundle/two`. Modify `hw-2-bundle/toy-app/pom.xml` so that the JavaMOP agent is attached to the test-running process to produce a `violation-counts` file.

4.4 (EXTRA CREDIT) Fix a false alarm in a JavaMOP specification (40 points)

`Collections_SynchronizedCollection.mop` (i.e., the `CSC` specification that we discussed several times in class) is violated when monitored against the provided `edu.cornell.cs6156.AppTest#testLockedCount`. Assuming that this violation is a false alarm (in the sense that we discussed in class [2]), create a new specification, `Corrected_Collections_SynchronizedCollection.mop` that fixes `Collections_SynchronizedCollection.mop`, does not generate a violation when running the provided `edu`

`.cornell.cs6156.AppTest#testLockedCount`, and still correctly generates a violation when running the provided `edu.cornell.cs6156.AppTest#testCount`. Add your `Corrected_Collections_SynchronizedCollection.mop` file to `hw-2-bundle/two`, monitor all three specifications that are now in `hw-2-bundle/two` to produce a `corrected-violation-counts` file. If you attempt this portion of the homework but are not able to complete it, document your experience in a file, `repair-experience.txt`.

4.5 Debriefing (10 points)

Answer the following questions in a file, `hw-2-bundle/debriefing.txt`. These questions are reused from CS 6114 [3].

1. How many hours did you spend on this homework?
2. Would you rate it as easy, moderate, or difficult?
3. How deeply do you feel you understand the material it covers (0%–100%)?
4. If you have any other comments, I would like to hear them! Please write them down or send email to legunsen@cornell.edu

4.6 Deliverable

Submit a single archive file with all the files created and edited during this homework.

```
$ cd hw-2-bundle/toy-app && mvn clean
$ cd hw-2-bundle/two && rm -rf *.aj *.rvm classes
$ tar czvf hw-2-submission.tgz hw-2-bundle
$ # upload hw-2-submission.tgz to CMS under "Homework 2"
```

References

[1] Oracle Inc. *ReentrantLock (Java Platform SE 8)*. `https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/ReentrantLock.html`. 2020.

[2] Owolabi Legunsen, Wajih Ul Hassan, Xinyue Xu, Grigore Rosu, and Darko Marinov. "How good are the specs? A study of the bug-finding effectiveness of existing Java API specifications". In: *ASE*. 2016, pages 602–613.

[3] Nate Foster. *CS 6114 Homework 1*. `https://cornell-pl.github.io/cs6114/homework01.html`. 2019.