Moka: Improving App Testing with Automated Mocking

1 Introduction

Nowadays, we use mobile apps for many of our daily activities, including reading the news, streaming content, and communicating with friends and family. Because some of these apps are used daily by millions of users, it is fundamental to thoroughly assess their quality and avoid serious failures due to undetected bugs. Testing has been shown to be very powerful in identifying bugs, but its effectiveness can be hindered by the flakiness of the tests. One common cause of flakiness is the non-deterministic behavior of the code under test, which is a frequent occurrence in mobile apps, due to the numerous interactions between apps and their environment. A common strategy to mitigate this kind of flakiness during testing is to manually create test mocks for specific test execution. However, manual mock creation is time consuming. Moreover, the generated mocks can typically only be used in the context of the specific tests for which they were created.

The goal of this project is to improve app testing by allowing, supporting, and partially automating the generation of smart test mocks. Specifically, we will develop and implement MOKA, a family of techniques that developers can use to collect, generalize, and use test mocks for manually and automatically generated tests. Intuitively, given an app under test (AUT), our techniques will (1) use record-and-reply techniques to collect mock data from the interactions between the AUT and the software environment, (2) generalize the collected data into smart test mocks, and (3) use these mocks to generate new tests. If successful, this research will provide unprecedented advantages to developers, who will be able to perform a more effective automated app testing while also mitigating the problem of test flakiness.

To achieve this goal, we will leverage our experience on app testing, program analysis, and record-and-replay techniques. In particular, we will leverage BARISTA [2], a testing technique based on record-and-replay and developed by two of the PIs. BARISTA, which allows developers to easily create platform-independent tests for mobile apps, is currently able to record and replay user actions and assertions, but it does not account for the interactions between the AUT and its environment. Handling these interactions and leveraging them to build mock objects is the starting point of this project.

2 Project Milestones and Outcomes

In this project, we will perform the following three research tasks:

Task 1 [months 1-4]: Collecting mock data. We will extend BARISTA so that it can instrument the AUT and collect I/O (e.g., file system, network, and sensors) data during app execution. We will design MOKA to be lightweight and minimally intrusive to prevent it from interfering with the behavior of the app. We will favor instrumenting the AUT over instrumenting the Android platform, so that the technique is portable across different versions of the operating system. MOKA will generate mock objects using the data recorded while running tests. It will then re-execute the tests multiple times to verify that the
generated mocks are effective at reducing/eliminating flakiness. To do so, MOKA will rely on the oracles within the tests and expect each execution to have the same outcome. If the flakiness comes from sources other than the ones we mocked, test may still manifest flaky behavior. If MOKA detects flakiness during the re-execution, it will mark the test accordingly.

**Task 2 [months 3-8]: Generalize and using test mocks.** We will define techniques to generalize mock data into test mocks. The design of the techniques will be along two dimensions. First, we will explore different ways to define the relationship between inputs and outputs in test mocks. Possible schemes include (i) using the same input and output relationship as seen during mock data collection and (ii) inferring an approximate relationship between inputs and outputs. Second, we will investigate the problem of how to select a test mock output when a given input leads to different outputs. Possible strategies include (i) considering the program stack when mock data was collected and (ii) selecting all outputs by conceptually forking the execution. In this task, we will also define an app instrumentation technique that enables developers to suitably use generated test mocks during automated test generation.

**Task 3 [months 9-12]: Experimentation on real-world apps.** We will perform rigorous experimentation to evaluate the effectiveness and efficiency of the techniques developed in the project. To this end, we will leverage our connections with industrial collaborators to assess the realistic nature of the assumptions underlying such experimentation, evaluate the techniques on real-world apps, and help the transition of the techniques to practice.

**Expected outcomes.** As we did in previous projects, we will publish the techniques developed in this project and their evaluation in top conferences and journals. We will also make the implementation of the techniques publicly available.

### 3 Team

**Mattia Fazzini**’s Ph.D. dissertation is on app testing, and he is the main author of BARISTA. **Alessandra Gorla** has worked on app analysis and created a platform for evaluating automated testing techniques for Android apps [1]. **Alessandro Orso** has worked on static and dynamic program analysis in multiple domains for over two decades. The three PIs collaborated on multiple projects and published several articles on app testing.

### References
