COMP4560 Project Report Draft

(Note: I’m not sure what references I need to use for either my Abstract or my introduction. I know I drew some material from a few sources for the introduction, including the main repo pages for FormatFuzzer and the hypothesis website. If I need to provide references for this draft, let me know and I’ll run through what I got from where)

# Abstract

Fuzzing is an effective tool to improve upon testing processes in software engineering, yet it sees little use amongst programmers working with higher-level languages such as Python. The tools that do exist, such as the open-source fuzzing tool FormatFuzzer, are not necessarily focussed on ease-of-use in a testing workflow. Hypothesis, however, is a property-based testing library for Python that aims to improve testing by using similar methods as fuzzing. Both FormatFuzzer and Hypothesis generate input data for some specification, however Hypothesis includes numerous additional features to improve its usability as a testing tool. Hypothesis currently lacks the versatility of FormatFuzzer, where FormatFuzzer can work on binary template files to produce many different types of input files. This project, titled Hypothesis-010, aims to improve the versatility of the Hypothesis library by using the Hypothesis features to enable more effective and efficient testing on many different types of binary template files. Hypothesis-010 will be tested for performance against other tools that exist for Fuzzing in Python, with the main comparison being against FormatFuzzer. If Hypothesis-010 is shown to be as effective in terms of performance as FormatFuzzer, but also includes the improved usability features of Hypothesis, this tool could vastly improve the viability of implementing fuzzing as a part of testing software coded in higher-level languages.

# Introduction

Hypothesis, the parent library of this project, is a property-based testing library for the Python language and seeks to streamline the testing process by enabling more effective and more efficient testing methods than current practice, which can be as inefficient as handwritten testing suites. Hypothesis does this by generating some input data to match some specification given by the programmer and testing that input data against the code to find instances where the code does not perform as desired. Accounting for edge cases and special cases of input data, Hypothesis tries to generate minimal canonical examples of input data that causes the code to fail in some way. This is done by using ‘shrinking’ strategies, that reduce the failing examples to the minimum possible example that still causes the same problems, with those minimal canonical examples becoming the useful output that testers can inspect to determine what went wrong.

This idea of automatically generating input data and aiming to cause problems in code for the purposes of testing is the key aspect of another area of testing, fuzzing. Fuzzing, like hypothesis, aims to produce incorrect performance in code by generating input data that causes the code to behave incorrectly despite the input data being recognised as valid. There are many fuzzing tools that already exist to facilitate implementing fuzzing, such as American Fuzzy Lop (AFL) and FormatFuzzer. FormatFuzzer is aimed at bringing fuzzing methods to Python, a higher-level language where the code presented to the programmer is quite abstracted away from the machine instructions. FormatFuzzer’s primary feature is to generate a fuzzing tool that can produce valid fuzzed files, which could then be used in the user’s testing process. It also has the capacity to record the decisions made while creating the valid fuzzed files and allows users to parse this valid fuzzed file to determine which of those key decisions made during the fuzzing process cause problems within their code. While FormatFuzzer is a satisfactory tool for fuzzing files, I believe that it lacks certain important features that would make it much more usable as a testing tool. These include the improved strategies of input generation and the capacity to report upon the failing examples that Hypothesis’s features provide.

Hypothesis-010 aims to bridge the gap, bringing Hypothesis data generation strategies to the field of binary template files that FormatFuzzer excels in. Taking in binary template files, Hypothesis-010 will determine the possible components of files that match the binary template and apply hypothesis strategies to each of those components to test them. This will result in a large composite strategy that can test all the components of a binary file and generate valid files that cause incorrect behaviour, as is the aim of the fuzzing tools that currently exist. With that composite strategy, Hypothesis-010 can then generate the desired output of minimal canonical examples of binary files, which are made more readily accessible to the user through Hypothesis recording the key details of those examples to reproduce them.

This paper details the software development process of Hypothesis-010 and compares the tool against other notable tools that already exist, namely FormatFuzzer.

(Note: This is where I’ve run out of steam, as I’m not quite sure on the measure of comparison here. I think the main comparison is in the examples produced by FormatFuzzer and Hypothesis-010, and whether Hypothesis-010 can produce minimal forms of the examples given by FormatFuzzer along with entirely new examples that FormatFuzzer doesn’t produce if the tool performs exceptionally. Leaving this note here as a reminder to ask about it next meeting!)