Testing Oracles and Automation

By Jordan Higgins

In this write-up I will explain my testing process, detail my usage of three different oracles, and relate my experience while implementing an automated testing environment using Javascript (Node JS specifically) and bash script.

I made use of javascript to set-up my testing environment, which consists of making 1,000 test files to run through my quadrilateral classifier. Of the 1,000 test files, 400 contain 6 integers, a single line, that make up a valid quadrilateral. I wrote a function in my script that will generate a randomized quadrilateral by first making a square, and then according to other randomized events, moves the points randomly in a randomly specified direction. This generates mostly quadrilaterals, trapezoids, squares, but also the occasional rectangle, kite, or rhombus. The first of the oracles I used to test my program involved shrinking the randomly generated quadrilateral, running the modified version of it through the program, and comparing it to the result of the original test. Theoretically, the shapes should be the same. If they are not, then I've found a bug in my program and the culprit file is identified.

The second oracle I implemented is similar to the first. For it, I created 4 different functions that caused semi-randomized occurrences of the 4 errors my program could encounter, invalid input, coinciding points, crossing lines, or 3 collinear lines. I dedicated 150 files to generating and then testing each one of these errors. I compared each file in the designated subset to a set of keys I generate, which is just four files with one line stating the type of error. This oracle hopes to ensure that my quadrilateral classifier is throwing the right errors.

The last oracle I implemented was compiling with an address sanitizer in the Make process, which will catch memory leaks and other related errors.

As a final note, I just want to mention that whoever runs my tests might have to have node js installed. I included the `#!/usr/bin/env node` at the top of my script, which tells the machine how to interpret the executable. It may or may not work with any mac computer, but installing node js will ensure that it does.