ASSIGNMENT 6 – ORACLES

JEREMY RODGERS

Over the past few days, I've been working on testing my Quadrilateral Classifier using randomized testing. In that time, I've learned how useful randomized testing can be. One scenario I ran into was a never-ending loop that would occur when I was generating the top left point for a Kite and the slope was 0. The loop occurred about once every 450 lines of code so needless to say, it would be very difficult to find that bug without some form of automated testing. Yet while automated testing can be useful, often times you are generating random inputs and as a result, it can be difficult to determine when the code is correct and when it is giving erroneous values. Oracles are a way to automatically find errors in a program. In my Quadrilateral Classifier, I made use of the following Oracles.

The compiler address sanitizer

The first oracle I used was the clang++ address sanitizer. The address sanitizer notifies a programmer when rule has been violated and therefore is an automatic indicator of errors. Specifically, the address sanitizer throws warnings on error like out-of-bound heap, stack and global access, use-after-free errors, use-after-return errors, memory leaks and more. These types of errors can be very difficult to detect so the address sanitizer can be a great tool to help identify problem code that could otherwise be very difficult to debug.

My Quadrilateral Builder

As outlined in our specification, my quadrilateral classifier takes in 3 points (the forth being implied as 0), and classifies the points as a quadrilateral type such as a square, a rhombus, a kite and so forth. To test that this implementation was correct, I built a program that does the inverse. My Test Builder takes in the name of a quadrilateral (such as a Kite) and randomly generates 3 points for the Quadrilateral Classifier to classify. To ensure that the answer is correct, the Test Builder also generates an answer key of what it intended to build. So if the Test Builder intended to build a square, it would print "square" to the key file and then print 4 0 4 4 0 4 to a test input file. The classifier then takes this input, classifies the quadrilateral as a square and produces an output of "square" to a separate output file. The output file and the key can then be compared for correctness. This was extremely useful because it allowed me to test all sorts of edge cases with each type of quadrilateral and it allowed me to still have 100% code coverage.

Assertions

Another useful Oracle I used to automatically detect errors was assertions. By adding assertions, I was able to automatically stop the program when certain conditions weren't met rather than continue the program with erroneous output. Assertions were useful everywhere,

but I found them especially handy when creating my test builder. Assertions allowed me to do things like assert that the four sides of a square were equal, or that the opposite slopes of a parallelogram were the same. I haven't used assertions a whole lot before now, but they will become a regular part of the programs I write. I found it so much easier to know exactly where the code was broken. It made it much easier to go in and fix it.