Erik Poole

CS 6015

Assignment 6 - Oracles

I employed four key oracles in this stage of our randomized testing – an address sanitizer, random string inputs of varied lengths composed of random characters, randomized valid integer inputs, and finally randomized inputs that should represent every shape we need to classify.

The first oracle, an address sanitizer, was used throughout development of the software and is one of our checks every time the program is compiled and run now. Identifying when the program might be reading out of its expected bounds or accessing invalid pointers is difficult to detect and can be very pernicious – oftentimes showing up only inconsistently. We're fortunate to have access to such a powerful tool at all stages of development. This was not difficult to implement from a developer's viewpoint, but it may be the most effective oracle on this list so there's hardly a reason not to include it.

Feeding the program completely random inputs is an excellent brute force method of testing and it's actually the only one of the four oracles implemented that showed me a bug that I hadn't yet encountered. Making sure that your program is impervious to erroneous input makes it significantly more robust and (perhaps even more importantly) making sure that the program's output isn't misleading when exposed to bad input is also important. The new error that I caught during this testing was reading blank files. I had previously accounted for this in input fed in via xCode's console, but when fed in a completely blank external fail the program wouldn't print any output instead of throwing an error. This is now corrected!

It's difficult to get hard data but randomized valid inputs, but they can help with informing suspicions and hunches. I get almost overwhelmingly quadrilaterals and "error 3" outputs from this test and that is more or less what I would expect. If enough tests were run you would probably start to see rarer quadrilateral types, but for the sample of 200 run for this assignment that output meets expectations.

Our last oracle is tested by providing randomized (but guaranteed) versions of all of the expected outputs. For shapes like squares and rectangles this is simply accomplished by scaling the lengths and widths, but for more complicated shapes like rhombi and trapezoids required shifts and changing angles as well. For the most part, these tests are relatively unintelligent and redundant, but they do a good job at confirming that my classifier works for at least the simple cases of each shape.