Monday, February 4, 2019

Fuzz

The 3 selected oracles are:

- Null space transformations. To test this, I first examined how the standard C++ library handled the std::stoi routine with arguments involving (), extra whitespace, and decimal representations that loose nothing in truncation when being converted to an int (I.e., 4.00 or 3.). All handled the transformations by converting the arguments to an int without any loss of fidelity in the numeric representation (in regards to our quadrilateral classifier). After viewing this behavior, tests were created to generate random quadrilaterals, but with one of the three characteristics above. To create extra whitespace, a random number of spaces (between 1 and 7) were put between each digit.

- Multiple versions. To test this, the python script runs 3 separate times using the same text files. Each iteration uses a different optimizer when building its profile. The different optimizations used were -O1, -O2, and -O3. At the end of the script, a comparison was done amongst the three coverage files to see if there was a difference in any of them, which there did not end up being.

- Function Inverse Pairs. While this one was difficult to create, due to the limitations of where the quadrilaterals could be located (E.g., no negatives), I turned to opposite of a quadrilateral. This didn't yield anything either as there is not necessarily an opposite shape. Instead, I built broken quadrilaterals. These were shapes comprised entirely of ASCII characters with any that might have been candidates for a quadrilateral (they had six number entries) were discarded. While these cannot be changed back to a quadrilateral, they were supposed to either break or trigger errors in the program. With a crash, I would know that the program did not handle all arguments the way it should.