Software Testing

CS-206

Project Report

Team Name: Defect Detectives

Team Members:

Raghu Ganapathy

Daiyaan Ahmed Shaik

March 8, 2023

1 Results

We performed an experiment on the 7 given benchmark programs and for each benchmark program, we created six test suites using the combinations of two coverage criteria (statements and branches) and three prioritization methods (random, total, and additional). The number of test cases in each test suite and the number of faults exposed by the test suite are recorded and tabulated below.
Note: For random prioritization techniques, we ran the code 10 times and recorded the mean and the standard deviation.

1.1 tcas

tcas		
Test Suite	**Size of the test suite**	**Number of faults exposed**
Base test suite	1590	41
Statement coverage with random test prioritization	5 ± 0.77	$\frac{8.66 \pm 4.5}{41}$
Statement coverage with total coverage prioritization	4	9/41
Statement coverage with additional coverage prioritization	4	9/41
Branch coverage with random test prioritization	11.9 ± 0.7	$\frac{14.66 \pm 0.94}{41}$
Branch coverage with total coverage prioritization	13	13/41
Branch coverage with additional coverage prioritization	11	13/41
Branch coverage with additional coverage prioritization	11	13/41

1.2 totinfo

totinfo		
Test Suite	**Size of the test suite**	**Number of faults exposed**
Base test suite	1026	21/23
Statement coverage with random test prioritization	$\frac{7.0 \pm 2.6}{41}$	$\frac{12.1 \pm 4.5}{23}$
Statement coverage with total coverage prioritization	5	12/23
Statement coverage with additional coverage prioritization	5	12/23
Branch coverage with random test prioritization	9	13/23
Branch coverage with total coverage prioritization	5	13/23
Branch coverage with additional coverage prioritization	5	12/23

1.3 schedule

schedule		
Test Suite	**Size of the test suite**	**Number of faults exposed**
Base test suite	2634	9/9
Statement coverage with random test prioritization	6.4 ± 1.36	$\frac{3.3 \pm 1.24}{9}$
Statement coverage with total coverage prioritization	3	2/9
Statement coverage with additional coverage prioritization	3	2/9
Branch coverage with random test prioritization	10.1 ± 1	$\frac{3 \pm 1.4}{9}$
Branch coverage with total coverage prioritization	8	3/9
Branch coverage with additional coverage prioritization	7	5/9

1.4 schedule2

schedule2		
Test Suite	**Size of the test suite**	**Number of faults exposed**
Base test suite	2679	9/9
Statement coverage with random test prioritization	3.7 ± 1.27	$\frac{1.6 \pm 0.47}{9}$
Statement coverage with total coverage prioritization	1	3/9
Statement coverage with additional coverage prioritization	1	3/9
Branch coverage with random test prioritization	10.7 ± 1.8	2/9
Branch coverage with total coverage prioritization	7	5/9
Branch coverage with additional coverage prioritization	5	2/9

1.5 printtokens

printtokens		
Test Suite	**Size of the test suite**	**Number of faults exposed**
Base test suite	4072	7/7
Statement coverage with random test prioritization	15.5 ± 3.07	$\frac{4.6 \pm 0.94}{7}$
Statement coverage with total coverage prioritization	6	6/7
Statement coverage with additional coverage prioritization	5	5/7
Branch coverage with random test prioritization	16.5 ± 1.8	$\frac{4.3 \pm 1.24}{7}$
Branch coverage with total coverage prioritization	7	5/7
Branch coverage with additional coverage prioritization	6	4/7

1.6 printtokens2

printtokens2		
Test Suite	**Size of the test suite**	**Number of faults exposed**
Base test suite	4057	9/9
Statement coverage with random test prioritization	11.5 ± 1.36	$\frac{7 \pm 0.8}{9}$
Statement coverage with total coverage prioritization	4	6/9
Statement coverage with additional coverage prioritization	4	6/9
Branch coverage with random test prioritization	15.6 ± 3.03	$\frac{6 \pm 0.8}{9}$
Branch coverage with total coverage prioritization	6	7/9
Branch coverage with additional coverage prioritization	4	6/9

1.7 replace

replace		
Test Suite	**Size of the test suite**	**Number of faults exposed**
Base test suite	5542	31/31
Statement coverage with random test prioritization	15.2 ± 1.83	$\frac{10.33 \pm 2.42}{31}$
Statement coverage with total coverage prioritization	12	10/31
Statement coverage with additional coverage prioritization	8	6/31
Branch coverage with random test prioritization	23.5 ± 2.3	$\frac{15.33 \pm 0.94}{31}$
Branch coverage with total coverage prioritization	20	19/31
Branch coverage with additional coverage prioritization	10	16/31

Benchmark	Maximum Statement Coverage	Maximum Branch Coverage
tcas	98.46%	92.42%
printtokens	94.97%	93.57%
printtokens2	100%	98.14%
totinfo	95.20%	89.77%
schedule	98.74%	95.45%
schedule2	100%	94.31%
replace	94.50%	93.33%

2 Observations

1. The sizes of the created test suites are significantly smaller than the original number of available test cases. For instance for the **replace** bench mark the average size of the test suites we report is 6, whereas the universe has 5542 test-cases. That is a 99.8% reduction in size.

2. The test suites with statement coverage are noted to be usually smaller than the test suites with branch coverage. This intuitively makes sense, because we know that complete branch coverage subsumes complete edge coverage. So we expect larger test suites when they are based on branch coverage than statement coverage.

3. Based on the faults are exposed by our test suites as compared to the total number of available faults, and as compared to the original test suite, we conclude that Total Coverage Prioritization based on branch Coverage generally detected more faults than the other generated test suites. Random Test Prioritization surprisingly gave us the best fault detection for **schedule** on one run, but did not consistently beat total or additional test prioritization for any given benchmark.

4. Some other interesting observations seen while developing the project were -

    - In Additional Test Prioritization, saw an opportunity for improving the performance of the overall runtime by using a double-ended queue instead of a dynamic array to store the remaining tests.
    - While attempting a multi-threaded implementation for fault detection, we ran into a lot of issues with file synchronization and decided to pivot back to detecting faults synchronously.