ECE 322
Lab Report 1

Arun Woosaree
XXXXXXX

October 6, 2019

Introduction

The purpose of this lab was to serve as a practical introduction to some more black box testing techniques. In this lab, the testing methods introduced were the Extreme Point Combination (EPC), and the Weak n x 1 strategy methods. We tested two programs written in Java using both of these testing strategies. The first application, named Drone takes in three arguments, and outputs either 'Success!', 'Failure', or an error message based on whether the arguments are integers ≥ 0, and their sum is less than $k = 100$. The second program, named RemoteCar takes in two arguments which represent points on a Cartesian plane, and outputs either 'Ok', 'Out of range', or an error message based on whether the point is on a circle of radius 1 about the origin. The idea for EPC testing is to identify the input domain limits, and produce all possible combinations of inputs with each of the input variables taking on a minimum value, slightly under minimum, a maximum value, and slightly over maximum value. One additional test case is added somewhere within the valid subdomain to generate a total of $4^n + 1$ test cases, where n is the dimension of inputs, or put more simply, the number of input variables. With the weak n x 1 strategy, we attempt to find linear boundaries of the problem using domain analysis. n points are selected on each linear boundary, where n is the number of input variables, and one additional point is chosen outside of each boundary. An additional point within the boundary is chosen if the boundary is open, and if the boundary is instead closed an additional point outside the boundary is chosen. In the end, with the weak n x 1 strategy, $b \times (n + 1)$ test cases are generated, where b is the number of linear boundaries, and n is the dimensionality, or the number of inputs.

Part 1 - Failure/Dirty Testing, Error Guessing

For task one in this lab, we had to be creative, as is the nature of Failure/Dirty testing, and error guessing. The purpose was to test the functionality of a calculator program, which was written in Java. A table of test cases was produced, checking for basic functionality, common errors. A few test cases were also made based on previous experience, which is also known as error guessing. Altogether, the test cases check for the following functionality:

1. whether the calculator buttons work
2. non-numerical input
3. mismatched brackets
4. order or operations (BEDMAS/PEMDAS)
5. large numbers
6. small numbers
7. incorrect syntax (e.g. 2++2)

The full list of test cases, along with the inputs and expected versus actual outputs can be found in Appendix. The test cases where the expected result does not match the actual result are highlighted in red. From experience, these test failures could be due to the following:

- numbers like 2^{512} are ridiculously big. It's likely that that the implementation was not designed to handle such large numbers.
- similarly, numbers like 2^{-512} are so small that the calculator interprets it as being the number 0
- entering nothing results in an output of 0. It's likely that a lone space is interpreted as a 0 by the calculator
- if there's a space between a number and the minus sign after it, the calculator outputs NaN. It could be that when parsing the input, the calculator may be interpreting it as two numbers with no operator in between, so it does not know what to do and returns an error (NaN)
- the calculator does not follow the correct order of operations. If there's a division and multiplication. (e.g. $80/4 * 5$), it appears to be doing the multiplication before the division, even though the multiplication should be done afterwards, since it is on the right hand side of the previous division operation.
- when multiple exponent (ˆ) operators are used in between two operands, the result is always one, instead of showing an error. This might be because the right half is evaluated as 0 by the calculator, and anything to the power of 0 is 1.
- when the first operand is missing for + the calculator seems to ignore it and treat it like a positive number. This is why the calculator says $2 + + + +2 = 4$

- when the first operand is missing for $*$, /, or ˆ, the calculator evaluates it as being equal to 0 instead of returning an error.
- with the exponent operator, the calculator tries to evaluate everything after the exponent. however, this violates the BEDMAS rule in cases where the exponent should be done before multiplication, division, addition, or subtraction. This results in errors like $2^1 + 1$ resulting in 4 instead of 3. This also compounds with the previous errors in some cases where an expression like $2^3 + 2^3$ is incorrectly evaluated as 2^5, since $3 + 2$ is evaluated first, which results in ˆ3 being evaluated as 1, and the expression is reduced to 2^5, resulting in the incorrect response.
- when numbers are only separated by a space, the calculator concatenates the digits. i.e. 1 2 becomes 12, instead of showing an error. when these numbers are wrapped by brackets, the calculator still concatenates them instead of multiplying. e.g. (2)(2) becomes 22 instead of 4. However, multiplying negative numbers using brackets e.g. (2)(-2) instead of the * operator results in a NaN error.
- the calculator has some rounding issues, which result in a loss of precision in the answer. Adding numbers like 2.000001+2.000002 result in a rounded answer instead of the exact answer.

Part 2 - Partition Testing

Task two of this lab involved partition-based testing of a triangle application. The purpose of this application is to take 3 space separated positive integers, each representing sides of a triangle, and the program is expected to tell the user whether the triangle is a scalene, isosceles, or equilateral triangle. The following equivalence classes were decided on for creating the test cases.

Triangle Equivalence Classes

From these equivalence classes, the following test cases were created:

Test cases for valid inputs

- 3 3 3 covers (1, 2, 3, 4, 7, 8)
- 4 4 5 covers (1, 2, 3, 5, 7, 8)
- 6 7 8 covers (1, 2, 3, 6, 7, 8)

Test cases for invalid inputs

- 1 2 covers (9)
- 3 4 5 6 covers (10)

- 7 8   9 covers (11)
- 8_7_6 covers (12)
- 1 −2 3 covers (13)
- 5 0 4 covers (14)
- 3 2 0.1 covers (15)
- 2 2 4 covers (16)
- 3 4 9 covers (17)
- not pressing Enter covers (18)

The results of these test cases, including expected versus actual output can be found in Appendix. Failed test cases are highlighted in red. There was one test case which failed.

- For the input where we have the case $a + b = c$, (2 2 4) in this case, the program tells us that it is a isosceles triangle instead of an invalid one. This is likely because the implementation checks validity by looking for $a + b > c$ instead of $a + b \geq c$. Similarly, with the inputs (1 2 3), the program tells us that it is a scalene triangle instead of showing an error.

Conclusion

In this lab, we were introduced to black-box testing. The techniques learned were dirty testing, error guessing, and partition-based testing. Dirty testing's strength seems to also be its weakness at the same time. That is, the test cases written are only limited by the tester's creativity. One disadvantage, however, is that one can generate a lot of extra test cases that are arguably unnecessary. That is, there is the possibility of having multiple tests which cover the same functionality of the program. This is a problem that using the partition-based testing method addressed. While equivalence testing allowed us to write significantly fewer test cases, it does not catch some unique cases. For example, if dirty testing was done instead, one might have tried the test inputs

$$1111111111\ 1111111111\ 1$$

, which returns ERROR: Invalid triangle instead of saying that it's an isosceles one. This error, and potentially others were not caught using partition-based testing. What is interesting, though is that the one test failure with inputs (2 2 4) for the Triangle program allowed for another error to be found, which is the case where an input like (1 2 3) which should cause an error instead returns 'Scalene'. Overall, it would not be fair to say that one testing method is universally better than the other, however, the tester would need to use their judgement and experience to choose an appropriate testing method for the software that they want to test.

Appendix A: Drone EPC Testing

Testid	description	x1	x2	x3	Expected	Actual
1	EPC permutation	100	100	100	Failure!	Failure!
2	EPC permutation	100	100	0	Failure!	Failure!
3	EPC permutation	100	100	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
4	EPC permutation	100	100	101	Failure!	Failure!
5	EPC permutation	100	0	100	Failure!	Failure!
6	EPC permutation	100	0	0	Success!	Success!
7	EPC permutation	100	0	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
8	EPC permutation	100	0	101	Failure!	Failure!
9	EPC permutation	100	-1	100	ERROR: Invald argument - negative value	Failure!
10	EPC permutation	100	-1	0	ERROR: Invald argument - negative value	Success!
11	EPC permutation	100	-1	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
12	EPC permutation	100	-1	101	ERROR: Invald argument - negative value	Failure!
13	EPC permutation	100	101	100	Failure!	Failure!
14	EPC permutation	100	101	0	Failure!	Failure!
15	EPC permutation	100	101	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
16	EPC permutation	100	101	101	Failure!	Failure!
17	EPC permutation	0	100	100	Failure!	Failure!
18	EPC permutation	0	100	0	Success!	Success!
19	EPC permutation	0	100	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
20	EPC permutation	0	100	101	Failure!	Failure!
21	EPC permutation	0	0	100	Success!	Success!
22	EPC permutation	0	0	0	Success!	Success!
23	EPC permutation	0	0	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
24	EPC permutation	0	0	101	Failure!	Failure!
25	EPC permutation	0	-1	100	ERROR: Invald argument - negative value	Success!
26	EPC permutation	0	-1	0	ERROR: Invald argument - negative value	Success!
27	EPC permutation	0	-1	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
28	EPC permutation	0	-1	101	ERROR: Invald argument - negative value	Success!
29	EPC permutation	0	101	100	Failure!	Failure!
30	EPC permutation	0	101	0	Failure!	Failure!
31	EPC permutation	0	101	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
32	EPC permutation	0	101	101	Failure!	Failure!
33	EPC permutation	-1	100	100	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
34	EPC permutation	-1	100	0	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
35	EPC permutation	-1	100	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
36	EPC permutation	-1	100	101	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
37	EPC permutation	-1	0	100	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
38	EPC permutation	-1	0	0	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
39	EPC permutation	-1	0	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
40	EPC permutation	-1	0	101	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
41	EPC permutation	-1	-1	100	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
42	EPC permutation	-1	-1	0	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
43	EPC permutation	-1	-1	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
44	EPC permutation	-1	-1	101	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
45	EPC permutation	-1	101	100	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
46	EPC permutation	-1	101	0	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value

47	EPC permutation	-1	101	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
48	EPC permutation	-1	101	101	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
49	EPC permutation	101	100	100	Failure!	Failure!
50	EPC permutation	101	100	0	Failure!	Failure!
51	EPC permutation	101	100	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
52	EPC permutation	101	100	101	Failure!	Failure!
53	EPC permutation	101	0	100	Failure!	Failure!
54	EPC permutation	101	0	0	Failure!	Failure!
55	EPC permutation	101	0	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
56	EPC permutation	101	0	101	Failure!	Failure!
57	EPC permutation	101	-1	100	ERROR: Invald argument - negative value	Failure!
58	EPC permutation	101	-1	0	ERROR: Invald argument - negative value	Success!
59	EPC permutation	101	-1	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
60	EPC permutation	101	-1	101	ERROR: Invald argument - negative value	Failure!
61	EPC permutation	101	101	100	Failure!	Failure!
62	EPC permutation	101	101	0	Failure!	Failure!
63	EPC permutation	101	101	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
64	EPC permutation	101	101	101	Failure!	Failure!
65	one valid test case	10	20	30	Success!	Success!

Appendix B: Drone Weak n x 1 Strategy

Testid	description	x1	x2	x3	Expected	Actual
1	x1+x2+x3=100	10	40	50	Success!	Success!
2	x1+x2+x3=100	35	45	20	Success!	Success!
3	x1+x2+x3=100	69	20	11	Success!	Success!
4	just outside of x1+x2+x3 = 100 boundary	33	34	34	Failure!	Failure!
5	x1=0, x2 + x3 < 100	0	20	30	Success!	Success!
6	x1=0, x2 + x3 < 100	0	50	20	Success!	Success!
7	x1=0, x2 + x3 < 100	0	21	36	Success!	Success!
8	just outside of x1=0 boundary	-1	25	62	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
9	x2=0, x2 + x3 < 100	32	0	16	Success!	Success!
10	x2=0, x2 + x3 < 100	48	0	25	Success!	Success!
11	x2=0, x2 + x3 < 100	42	0	14	Success!	Success!
12	just outside of x2=0 boundary	21	-1	42	ERROR: Invald argument - negative value	Success!
13	x3=0, x2 + x3 < 100	6	69	0	Success!	Success!
14	x3=0, x2 + x3 < 100	72	5	0	Success!	Success!
15	x3=0, x2 + x3 < 100	65	12	0	Success!	Success!
16	just outside of x3=0 boundary	45	7	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
17	additional test case inside the boundary	10	20	30	Success!	Success!

Appendix C: RemoteCar EPC Testing

Testid	description	x	y	Expected	Actual
1	EPC permuation	-0.1	-0.1	Ok.	Ok.
2	EPC permuation	-0.1	0	Ok.	Ok.
3	EPC permuation	-0.1	1	Out of range!	Out of range!
4	EPC permuation	-0.1	1.1	Out of range!	Out of range!
5	EPC permuation	0	-0.1	Ok.	Ok.
6	EPC permuation	0	0	Ok.	Ok.
7	EPC permuation	0	1	Ok.	Ok.
8	EPC permuation	0	1.1	Out of range!	Out of range!
9	EPC permuation	1	-0.1	Out of range!	Out of range!
10	EPC permuation	1	0	Ok.	Ok.
11	EPC permuation	1	1	Out of range!	Out of range!
12	EPC permuation	1	1.1	Out of range!	Out of range!
13	EPC permuation	1.1	-0.1	Out of range!	Out of range!
14	EPC permuation	1.1	0	Out of range!	Out of range!
15	EPC permuation	1.1	1	Out of range!	Out of range!
16	EPC permuation	1.1	1.1	Out of range!	Out of range!
17	one valid test case	0	0	Ok.	Ok.

Appendix D: RemoteCar Weak n x 1 Strategy

Testid	description	x	y	Expected	Actual
1	y=1-x boundary	0.9	0.1	Ok.	Ok.
2	y=1-x boundary	0.1	0.9	Ok.	Ok.
3	just outside of y=1-x boundary	0.5	0.6	Out of range!	Ok.
4	y=1+x boundary	-0.1	0.9	Ok.	Ok.
5	y=1+x boundary	-0.9	0.1	Ok.	Ok.
6	just outside of y=1+x boundary	-0.5	0.6	Out of range!	Ok.
7	y=-(1+x) boundary	-0.9	-0.1	Ok.	Ok.
8	y=-(1+x) boundary	-0.1	-0.9	Ok.	Ok.
9	just outside of y=-(1+x) boundary	-0.5	-0.6	Out of range!	Ok.
10	y=x-1 boundary	0.1	-0.9	Ok.	Ok.
11	y=x-1 boundary	0.9	-0.1	Ok.	Ok.
12	just outside of y=x-1 boundary	0.5	-0.6	Out of range!	Ok.
13	additional test case inside the boundary	0.3	0.7	Ok.	Ok.