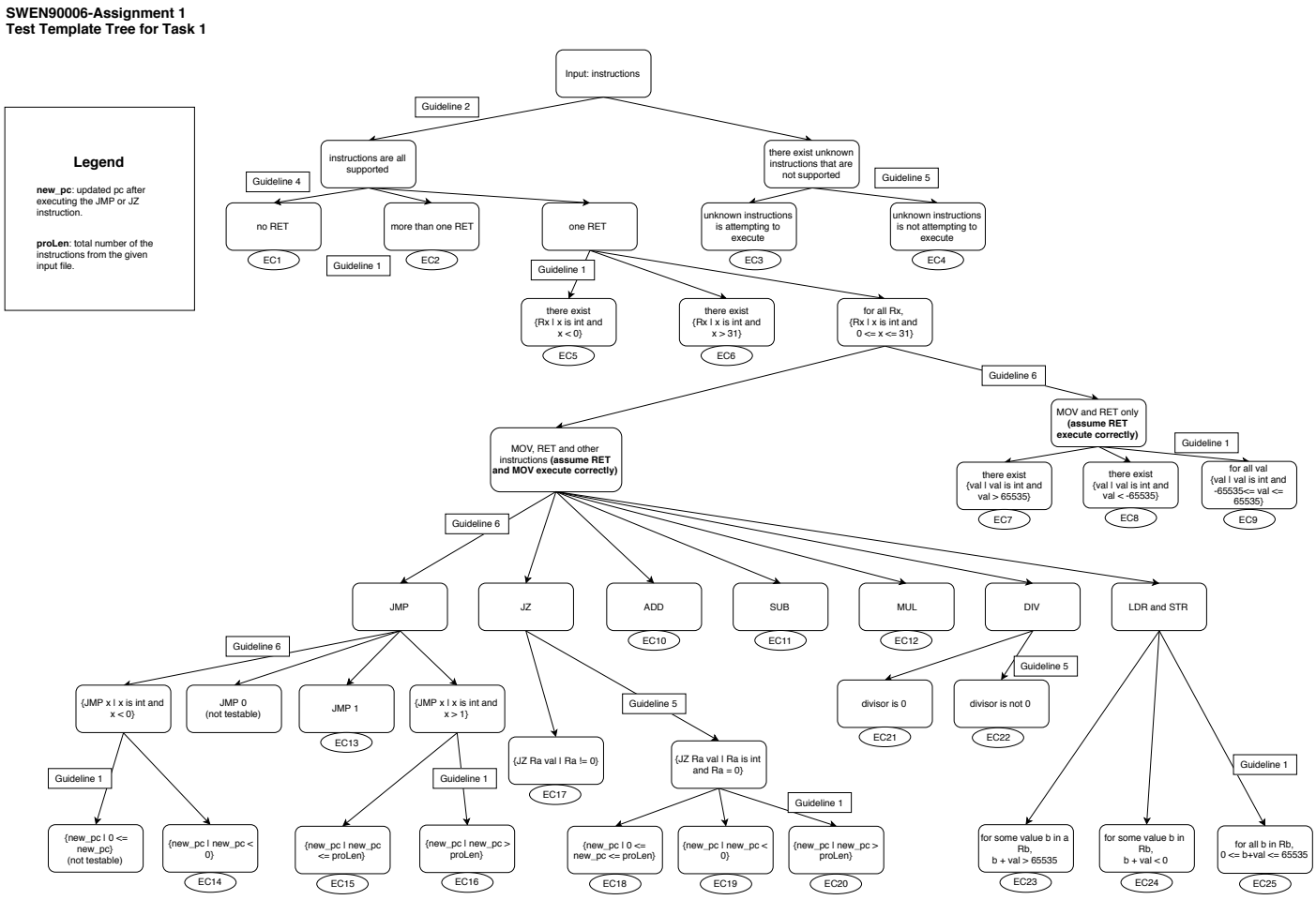
SWEN90006 Software Testing and Reliability

Assignment 1
Semester 2 – 2018

Name redacted

Task 1

Test Template Tree



Question

Do your set of equivalence classes cover the input space?

Answer

Yes

Justification

My 25 equivalent classes do cover the input space because each leaf node derived does not overlap with one another and each breakdown covers its parent node by following the guidelines.

I do not need to cover the case where there exist instructions that contain syntactical mistake because all instructions are assumed to have no syntax errors. Similarly, I do not need to consider the case where the input instruction is empty because it is assumed to have at least one instruction.

However, I need to consider the case that dividend is divided by 0 under “DIV” instruction. Also, I need to use

LDR and STR together as one equivalent class (EC). This is because STR alone is just a method to store values into memory, and I need to retrieve the values back so that I can validate its correctness.

More importantly, I need to make assumptions when testing instructions {JZ, JMP, ADD, SUB, MUL, DIV, STR, LDR}. The assumptions are MOV and RET are executing correctly so that I can use these two instructions to examine the behaviors of the above-mentioned instructions. In addition, I need to assume RET executes correctly to test MOV's behavior in EC7, EC8 and EC9.

Last but not the least, test cases become untestable if there is an infinite loop and that can be an expected behavior for the Machine class. Then we should try to avoid such cases and could choose to consider them as syntax errors. If the program enters an infinite loop and cannot be finished or returned, we could consider it as not passing the partitioning test cases.

Globally, the whole leaf nodes cover the input space by ensuring the properties of non-overlapping and coverage for every partition.

Task 3

Using Boundary-value analysis, we attempt to select test cases near the boundary by selecting test cases on and around the boundary using the 4 guidelines provided in the notes. In this task, I continue to consider the test cases untestable if they contain infinite loops. It is because there is no way to check the Machine object's state using, for example, "getCount()" method, while the "execute" method is in the loop.

The Boundary-value analysis on remaining equivalent classes will be shown in the following table. Note that

1. "**EC**" refers to "Equivalent Class",
2. "**IIE**" refers to "InvalidInstructionException",
3. "**NRE**" refers to "NoReturnValueException" and
4. "**IL**" refers to "Infinite loop".

EC	Boundary	Boundary Type	Test case selections	Actual instructions generated	Expected Output
1	Number(RET) = 0	Strict Equality	Using Guideline 1: 1. On point: 0 2. Off point: -1(untestable) 3. Off point: 1	1. MOV R0 1 2. RET R0	1. NRE 2. 0
2	Number(RET) > 1	Inequality, open	Using Guideline 2: 1. On point: 1 2. Off point: 2 Using Guideline 4: On point will generate identical test case as EC1, so we do not select this on point.	1. RET R0 MOV R1 1 RET R1	1. 0
3	Number(unknow instructions execute) > 0	Inequality, open	Using Guideline 2: 1. On point: 0 2. Off point: 1	1. RET R0 FOO 1 2. FOO 1 RET R0	1. 0 2. IIE
4	Number(unknow instructions execute) = 0	Strict Equality	Using Guideline 1: 1. On point: 0 2. Off point: -1(untestable) 3. Off point: 1 Using Guideline 4: Testable on point and off point will generate identical test cases as EC3, so we do not select these on point and off point.	NA	NA
5	{Rx | x is integer and x < 0}	Inequality, open	Using Guideline 2: 1. On point: 0 2. Off point: -1	1. RET R0 2. RET R-1	1. 0 2. IIE
6	{Rx | x is integer and x > 31}	Inequality, open	Using Guideline 2: 1. On point: 31 2. Off point: 32	1. RET R31 2. RET R32	1. 0 2. IIE
7	{val | val is integer and val > 65535}	Inequality, open	Using Guideline 2: 1. On point: 65535	1. MOV R0 65535 RET R0	1. 65535 2. IIE

			2. Off point: 65536	2. MOV R0 65536 RET R0	
8	{val | val is integer and val < -65535}	Inequality, open	Using Guideline 2: 1. On point: -65535 2. Off point: -65536	1. MOV R0 -65535 RET R0 2. MOV R0 -65536 RET R0	1. -65535 2. IIE
9	{val | val is integer and -65535 <= val <= 65535}	Inequality, closed	Using Guideline 4: Do not select identical tests for adjacent ECs (8, 9, 10). 1. EC9: -65535 <= val and EC10: val < -65535 have identical tests. 2. EC8: val > 65535 and EC10: val <= 65535 have identical tests.	NA	NA
10	Number(ADD) > 0	Inequality, open	Using Guideline 2: 1. On point: 0 2. Off point: 1 Using Guideline 4: On point for this EC will generate identical test case as EC7 and EC8 (combination of MOV and RET only), so we do not select this on point.	1. MOV R0 1 MOV R1 2 ADD R2 R1 R0 RET R2	1. 3
11	Number(SUB) > 0	Inequality, open	Using Guideline 2: 1. On point: 0 2. Off point: 1 Using Guideline 4: Due to similar reason as EC10, we do not select this on point.	1. MOV R0 -1 MOV R1 2 SUB R2 R1 R0 RET R2	1. 3
12	Number(MUL) > 0	Inequality, open	Using Guideline 2: 1. On point: 0 2. Off point: 1 Using Guideline 4: Due to similar reason as EC10, we do not select this on point.	1. MOV R0 -1 MOV R1 -3 MUL R2 R1 R0 RET R2	1. 3
13	For some JMP x, {x | x = 1}	Strict Equality	Using Guideline 1: 1. On point: 1 2. Off point: 0 (IL, untestable) 3. Off point: 2	1. JMP 1 MOV R0 1 RET R0 2. JMP 2 MOV R0 1 RET R0	1. 1 2. 0

14	For some JMP x and n_pc which is updated 'pc' after executing this JMP, {JMP x, n_pc | x is int and x < 0 and n_pc < 0}	Inequality, open	Using Guideline 2: For x < 0: 1. On point: 0 (IL, untestable) 2. Off point: -1 For n_pc < 0: 3. On point: 0 (IL, untestable) 4. Off point: -1	1. JMP -1 MOV R0 1 RET R0	1. NRE
15	For some JMP x and n_pc which is updated 'pc' after executing this JMP, {JMP x, n_pc | x is int and x > 1 and n_pc <= progLength}	Inequality: x > 1: open n_pc <= progLength: closed	Using Guideline 2: For x > 1: 1. On point: 1 2. Off point: 2 For n_pc<=progLength: 3. On point: progLength 4. Off point: progLength + 1 Using Guideline 4: I need to avoid generating identical test cases as those in EC13, so there are only two test cases created.	1. MOV R0 1 JMP 1 RET R0 2. MOV R0 1 JMP 2 RET R0	1. 1 2. NRE
16	For some JMP x and n_pc which is updated 'pc' after executing this JMP, {JMP x, n_pc | x is int and x > 1 and n_pc > progLength}	Inequality, open	Using Guideline 4: Do not select identical tests for adjacent ECs (15, 16).	NA	NA
17	{JZ Ra val | Ra is int and Ra != 0}	Inequality: Ra > 0: open Ra < 0: open	Using Guideline 2&4: 1. On point: 0 2. Off point: -1, 1	1. JZ R0 2 MOV R1 1 RET R1 2. MOV R0 -1 JZ R0 2 RET R0 3. MOV R0 1 JZ R0 2 RET R0	1. 0 2. -1 3. 1
18	For some n_pc which is updated 'pc' after executing this JZ, {JZ Ra val, n_pc | Ra is int and Ra = 0 and 0 <= n_pc <= progLength}	Strict Equality: Ra = 0 Inequality: closed	Using Guideline 1: For Ra = 0: 1. On point: 0 2. Off point: -1 ,1 Using Guideline 2: For 0 <= n_pc: 1. On point: 0 (IL, untestable)	1. JZ R0 -1 MOV R0 1 RET R0 2. JZ R0 3 MOV R0 1 RET R0	1. NRE 2. NRE 3. -1 4. -1 5. 1 6. 1

			2. Off point: -1 For n_pc <= progLength: 3. On point: progLength 4. Off point: progLength + 1 Using Guideline 4: I need to avoid generating identical test cases as those in EC17, so there are only 6 test cases created.	3. MOV R0 -1 JZ R0 -2 RET R0 4. MOV R0 -1 JZ R0 1 RET R0 5. MOV R0 1 JZ R0 -2 RET R0 6. MOV R0 1 JZ R0 1 RET R0	
19	For some n_pc which is updated 'pc' after executing this JZ, {JZ Ra val, n_pc | Ra is int and Ra = 0 and 0 > n_pc}	Strict Equality: Ra = 0 Inequality: 0 > n_pc: open	Using Guideline 4: EC19 and EC18 are adjacent equivalence classes, and EC19 will generate identical test cases that exists in EC18.	NA	NA
20	For some n_pc which is updated 'pc' after executing this JZ, {JZ Ra val, n_pc | Ra is int and Ra = 0 and n_pc > progLength}	Strict Equality: Ra = 0 Inequality: n_pc > progLength	Using Guideline 4: EC20 and EC18 are adjacent equivalence classes, and EC20 will generate identical test cases that exists in EC18.	NA	NA
21	{DIV Ra Rb Rc | Rc = 0}	Strict Equality	Using Guideline 1: 1. On point: 0 2. Off point: -1, 1	1. MOV R1 1 DIV R2 R1 R0 RET R2 2. MOV R1 1 MOV R0 1 DIV R2 R1 R0 RET R2 3. MOV R1 1 MOV R0 -1 DIV R2 R1 R0 RET R2	1. 0 2. 1 3. -1
22	{DIV Ra Rb Rc | Rc != 0}	Inequality: Rc < 0: open Rc > 0: open	Using Guideline 4: EC22 and EC21 are adjacent equivalence classes, and EC22 will generate identical test cases that exists in EC21.	NA	NA
23	{STR Rb val Ra | Rb	Inequality,	Using Guideline 2:	1. MOV R0 65535	1. 1

	+ val > 65535}	open	1. On point: 65535 2. Off point: 65536	MOV R1 1 STR R0 0 R1 LDR R2 R0 0 RET R2 2. MOV R0 65535 MOV R1 1 STR R0 1 R1 LDR R2 R0 1 RET R2	2. 0
24	{STR Rb val Ra | Rb + val < 0}	Inequality, open	Using Guideline 2: 1. On point: 0 2. Off point: -1	1. MOV R1 1 STR R0 0 R1 LDR R2 R0 0 RET R2 2. MOV R1 1 STR R0 -1 R1 LDR R2 R0 -1 RET R2	1. 1 2. 0
25	{STR Rb val Ra | 0 <= Rb + val <= 65535}	Inequality: closed	Using Guideline 4: EC25, EC24 and EC23 are adjacent equivalence classes, and EC25 will generate identical test cases that exists in EC23 and EC24.	NA	NA

Total test cases: 37

Task 5

Using Multi-condition Converge on only the “execute” method in the Machine class:

Now, here are listing the conditions only in “execute” method and I label them as {A-Z}:

A: while(true)

B: if (pc < 0 || pc >= progLength)

C: if (inst.equals(""))

D: if (toks.length < 2)

E: if (toks[0].equals(INSTRUCTION_ADD))

F: if (toks.length != 4)

G: else if (toks[0].equals(INSTRUCTION_SUBTRACT))

H: if (toks.length != 4)

I: else if (toks[0].equals(INSTRUCTION_MULT))

J: if (toks.length != 4)

K: else if (toks[0].equals(INSTRUCTION_DIVIDE))

L: if (toks.length != 4)

M: else if (toks[0].equals(INSTRUCTION_RETURN))

N: else if (toks[0].equals(INSTRUCTION_LOAD))

O: if (toks.length != 4)

P: else if (toks[0].equals(INSTRUCTION_STORE))

Q: if (toks.length != 4)

R: else if (toks[0].equals(INSTRUCTION_MOVE))

S: if (toks.length != 3)

T: else if (toks[0].equals(INSTRUCTION_JUMP))

U: if (toks.length != 2)

V: else if (toks[0].equals(INSTRUCTION_JZ))

W: if (toks.length != 3)

X: if (regs[ra] == 0)

Y: else

Z: else

Since A has a condition “true” which can never be false, only one edge is able to leave from node A. Hence, A cannot be considered as a branch according to the definition of branch in the notes.

Therefore, the number of total test objects are $25 \times 2 = 50$ in this case as there are 25 branches.

1. Coverage Score of Partitioning Tests:

EC Test Case	True	False
1	RB	BCDEGIKMNPS
2	RM	BCDEGIKMNPSBCDEGIK
3	Z	BCDEGIKMNPRTV
4	M	BCDEGIK
5	M	BCDEGIK
6	M	BCDEGIK
7	R	BCDEGIKMNPS
8	R	BCDEGIKMNPS
9	RM	BCDEGIKMNPBCDEGIK
10	RREM	BCDEGIKMNPSBCDEGIKMNPSBCDFBCDEGIK

11	RRGM	BCDEGIKMNPSBCDEGIKMNPSBCDEHBCDEGIK
12	RRIM	BCDEGIKMNPSBCDEGIKMNPSBCDEGJBCDEGIK
13	TRM	BCDEGIKMNPRUBCDEGIKMNPSBCDEGIK
14	RT	BCDEGIKMNPSBCDEGIKMNPRU
15	TM	BCDEGIKMNPRUBCDEGIK
16	TV	BCDEGIKMNPRU
17	RVYM	BCDEGIKMNPSBCDEGIKMNPRTWXBCDEGIK
18	VXRM	BCDEGIKMNPRTWYBCDEGIKMNPSBCDEGIK
19	VXB	BCDEGIKMNPRTW
20	VXB	BCDEGIKMNPRTW
21	RRKM	BCDEGIKMNPSBCDEGIKMNPSBCDEGILBCDEGIK
22	RRKM	BCDEGIKMNPSBCDEGIKMNPSBCDEGILBCDEGIK
23	RRPNM	BCDEGIKMNPSBCDEGIKMNPSBCDEGIKMNQBCDEGIKMOBCDEGIK
24	RRPNM	BCDEGIKMNPSBCDEGIKMNPSBCDEGIKMNQBCDEGIKMOBCDEGIK
25	RRPNM	BCDEGIKMNPSBCDEGIKMNPSBCDEGIKMNQBCDEGIKMOBCDEGIK

After running all test cases from PartitioningTests, the true test objects met include {B,E,G,I,K,M,N,P,R,T,V,X,Y,Z}, and the false test objects met include {B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y}. Hence the coverage score =

$$\frac{number\ of\ test\ objects\ met}{total\ number\ of\ test\ objects} = \frac{14+24}{50} \times 100\% = 76\%$$

2. Coverage Score of Boundary Tests:

EC Test Case	True	False
1.1	RB	BCDEGIKMNPS
1.2	M	BCDEGIK
2.1	M	BCDEGIK
3.1	M	BCDEGIK
3.2	Z	BCDEGIKMNPRTV
5.1	M	BCDEGIK
5.2	M	BCDEGIK
6.1	M	BCDEGIK
6.2	M	BCDEGIK
7.1	RM	BCDEGIKMNPSBCDEGIK
7.2	R	BCDEGIKMNPS
8.1	RM	BCDEGIKMNPSBCDEGIK
8.2	R	BCDEGIKMNPS
10.1	RREM	BCDEGIKMNPSBCDEGIKMNPSBCDFBCDEGIK
11.1	RRGM	BCDEGIKMNPSBCDEGIKMNPSBCDEHBCDEGIK
12.1	RRIM	BCDEGIKMNPSBCDEGIKMNPSBCDEGJBCDEGIK
13.1	TRM	BCDEGIKMNPRUBCDEGIKMNPSBCDEGIK
13.2	TM	BCDEGIKMNPRUBCDEGIK
14.1	T	BCDEGIKMNPRU
15.1	RTM	BCDEGIKMNPSBCDEGIKMNPRUBCDEGIK
15.2	RT	BCDEGIKMNPSBCDEGIKMNPRU
17.1	VXM	BCDEGIKMNPRTWBCDEGIK
17.2	RVYM	BCDEGIKMNPSBCDEGIKMNPRTWXBCDEGIK
17.3	RVYM	BCDEGIKMNPSBCDEGIKMNPRTWXBCDEGIK

18.1	VX	BCDEGIKMNPRTW
18.2	VX	BCDEGIKMNPRTW
18.3	RVYM	BCDEGIKMNPSBCDEGIKMNPRTWXBCDEGIK
18.4	RVYM	BCDEGIKMNPSBCDEGIKMNPRTWXBCDEGIK
18.5	RVYM	BCDEGIKMNPSBCDEGIKMNPRTWXBCDEGIK
18.6	RVYM	BCDEGIKMNPSBCDEGIKMNPRTWXBCDEGIK
21.1	RKM	BCDEGIKMNPSBCDEGILBCDEGIK
21.2	RRKM	BCDEGIKMNPSBCDEGIKMNPSBCDEGILBCDEGIK
21.3	RRKM	BCDEGIKMNPSBCDEGIKMNPSBCDEGILBCDEGIK
23.1	RRPNM	BCDEGIKMNPSBCDEGIKMNPSBCDEGIKMNQBCDEGIKMOBCDEGIK
23.2	RRPNM	BCDEGIKMNPSBCDEGIKMNPSBCDEGIKMNQBCDEGIKMOBCDEGIK
24.1	RPNM	BCDEGIKMNPSBCDEGIKMNQBCDEGIKMOBCDEGIK
24.2	RPNM	BCDEGIKMNPSBCDEGIKMNQBCDEGIKMOBCDEGIK

After running all test cases from BoundaryTests, the true test objects met include {B,E,G,I,K,M,N,P,R,T,V,X,Y,Z}, and the false test objects met include {B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y}. Hence the coverage score =

$$\frac{number\ of\ test\ objects\ met}{total\ number\ of\ test\ objects} = \frac{14+24}{50} \times 100\% = 76\%$$

Question 7

First of all, the coverage scores on “execute” method in the Machine class for two sets of test cases are the same. This can be because the equivalent classes generated from the leaf nodes of the test template tree are non-generalized and non-overlapping, which improves the performance of equivalent partitioning test cases. It makes the multi-condition coverage similar or even the same by only considering the “execute” method. However, if we take a closer look into the other methods such as “do_move”, the coverage may then differ. Based on the results from task 5 alone, we cannot conclude which one is better than the other. Also, both sets generate test cases based on the same ECs and the ECs from the test template tree covers the input domain as explained in Task 1. Both sets of test cases are able to cover the output domain including throwing all available exceptions (InvalidInstructionException, NoReturnValueException), return a default register’s value (=0) and return a register’s assigned value as well. Therefore, they are both considered to have the coverage of input/output domain.

On the other hand, in terms of killing mutants, the BoundaryTests are able to kill all five mutants I created for Task 6. On the other hand, the PartitioningTests is only able to kill ‘mutant 2’. Therefore, the BoundaryTests are much more effective than the PartitioningTests.

In addition, according to the subject notes, boundary-value analysis can be considered as a ‘refinement’ for the equivalent partitioning and the results are consistent with this statement. It contains total of 37 test cases to target the values on and around the EC’s boundaries, thus perform testing better.

In conclusion, the set of test cases for boundary-value analysis is more effective than that for equivalent partitioning.