HOMEWORK ASSIGNMENT: CHAPTER 12

**Question 12.1:**

Provide an explanation of MC/DC vs. compound condition adequacy with regard to the compound expression: for ( n=0 ; n<max\_size && (c=getc(yyin))!=EOF && c!=’\n’ ; n++)

**Response:**

By determining which basic conditions within the above compound conditional expression have an effect on the overall conditional expression’s evaluation, we can simplify the number of test cases required to ensure statement and branch coverage that follows the program’s evaluation of it. MC/DC is the type of coverage-adequacy extension that ensures the program is thorough, in regard to its execution sequence, with minimal burden to the test suite developers, in other words.

If the compound condition expression is broken into basic conditions: a) n<max\_size …. C) c!=’\n’, we might derive a compound condition coverage table that consists of:

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Basic condition A:  N < Max\_size | Basic condition B:  c=getc(yyin)!=EOF | Basic condition C:  c!=’\n’ |
| 1 | T | T | T |
| 2 | T | T | F |
| 3 | T | F | F |
| 4 | F | T | T |
| 5 | T | F | T |
| 6 | F | F | F |

The worst case for a compound conditional expression with three basic conditions being 2^3=8 test cases, (\*we may be leaving out a few cases, with the above table but due to the configuration of short-circuit operators, it seems there is a worst-case of 6 possible test cases per suite in this compound expression statement). However, if we assess evaluating the compound expression statement with respect to how short-circuit evaluation impacts it, we can minimize the potential size of the test suite and arrive at a smaller test suite, such that not only are the red shaded test cases deemed unnecessary, but also evaluation of the basic condition ‘a’, altogether.

For example, with compound condition coverage, we might need a test suite T0 {*max\_size , c value*}consisting of: T0={ 1, ‘%’ , 1, ‘\n’ , 1, ’--‘ , 0, ‘%’ , 1, ‘EOF’ , 0, ‘--’ }.

\*Because the compound condition adequacy criterion does not factor out needed test cases respective of the impact of short-circuit evaluation, the red-shaded rows (rows where conditions b,c can both be false – an impossibility in the real-world!) indicate the possibility where fewer additional test cases could be needed are not eliminated; nor are those cases which factor out basic condition ‘a’, meaning test cases 1 and 4 are needed. That is, where we might have had 6 test cases required for compound condition adequacy test suite, we might need a test suite with only 2 test cases for MC/DC coverage, as follows: T1{ 1, ‘\n’ , 1, ‘EOF’ }.

Since the basic conditions ‘b’ and ‘c’ are potentially mutually exclusive, in that they can BOTH be true at simultaneously (if for e.g: c=’%’….or !=”EOF” && !=’\n’). Therefore, we must evaluate the compound conditional expression with respect to this scenario – exclusive of evaluating the basic condition ‘a’ since it does not adversely or positively affect the overall outcome after, commutatively, with respect to conditions ‘b’ or ‘c’. So we are left with a worst-case suite size 2^1 other cases (the case, in fact,always when there only two basic conditions left to evaluate) irrespective of configuration of operators, mindful of precedence factor. Where we might have more cases to consider with || operator, especially when ()’s are part of the scenario, in this case with &&, two operands (2 basic conditions)…and no ()’s, we do have a suite of two test cases, which is T1.

**------------------------------------------------------------------------------------------------------------------------------------------**

**Question: 12.3:**

…..

**Response:**

To prove that N+1 test cases are required to satisfy the modified condition adequacy criterion for a predicate with N basic conditions, we consider the minimum basic conditions required which constitutes an executable predicate statement (by virtue of the Touring machine concept?)…a predicate with 1 basic condition, below:

Predicate: if/while(BC==true){}

The MC/DC definition states that to empirically show all possible paths are navigable, we can ensure that the basic conditions in the predicate can each have impact to the body of code that follows its evaluation will be reachable if for all other basic conditions in the compound expression, there is a test case where all the other values can be true or all of them false and the basic condition under consideration will change the path of execution depending on its evaluation result (ie: all other basic conditions are mutually exclusive to each other?)

Thus, above, if we consider each predict a result of counting the leaves in the binary decision tree/diagram, with 1 basic condition, we have:

Basic Condition

/ \

T F

/ \

RESULT🡪 T F