Analysis of MC/DC Coverage Percentage and Cyclometric Complexity for structured C Programs

Sangharatna Godbole
Assistant Professor
Nagpur Institute of Technology
Nagpur, Maharashtra
+91-7875490597
sanghu1790@gmail.com

Durga Prasad Mohapatra
Associate Professor
National Institute of Technology
Rourkela, Odhisa
+91-9437648628
durga@nitrkl.ac.in

ABSTRACT
Now a days, testing activity consumes about 60% of software development resources, so any concept aiming at reducing Software testing costs are likely to decrease the whole development costs. Proposed by NASA in 1994, the MC/DC criteria is a testing strategy required, among other practices, by the RTCA DO-178B. MC/DC is a white box testing criteria targeting at proving evidence that all clauses involved in a predicate can influence the predicate value in the required way. We automated the concept to identify predicates in structured C-Program using predicate identifier and Cyclometric Complexity Calculator. We automated the generation of number of test cases required to satisfy the MC/DC criteria using CREST Tool and evaluating MC/DC coverage percentage using coverage analyser.

Categories and Subject Descriptors
D.2.5 [SOFTWARE ENGINEERING]: Testing and Debugging, Testing tools (e.g., data generators, coverage testing)

General Terms

Keywords
Complexity Calculator, Coverage Analyser, CREST Tool

1. INTRODUCTION
In source code of automobile and airplane, a conditional expression consists of numerous boolean conditions and source code includes a large number of conditional expressions. MC/DC coverage is required effective way of generate test requirements which uses boolean conditions in conditional expressions because the tools supporting MC/DC coverage [2] for software testing are commercial products. The information regarding test requirements for MC/DC coverage is hidden. Therefore, this research which generates test requirements for MC/DC [3] coverage is required. As every thing is transparent in glass, like that visibility in all aspect for software shows the property of glass box testing.

CONCOLIC (CONcrete + symbOLIC) testing [12] (also known as dynamic symbolic execution [Tillman et al. [10] and white-box fuzzing [16]]) combines concrete dynamic analysis and static symbolic analysis to automatically generate test suite [11] to explore execution paths of a target program. Therefore, it is necessary to check whether CONCOLIC testing [12] can detect bugs in open source applications in a practical manner through case studies[17][7].

In this paper, we discussed about analysis of C programs through our proposed architecture. We used concolic tester CREST tool to generate test suite for structured C-program. We designed coverage analyser to calculate MC/DC coverage percentage after fetching generated test suite and C-program.

We designed complexity calculator to calculate cyclometric complexity for C-program after identifying predicates.

In second section of this paper, we discussed some basic concepts regarding the standard terminologies. In third section, we discussed some related research work in this area. Fourth section represents the concept of proposed work. Fifth section deals with analysis of MC/DC coverage percentage and cyclometric complexity and sixth section discusses about conclusion and future work.

2. BASIC CONCEPTS
In this section, we discussed some definitions and important terminologies [14][6].

1. Condition: The Boolean statement with no Boolean operator is known as condition or clause [8].

2. Predicate: The Boolean statement which consists of clauses and zero or more Boolean operators is called as predicate. A decision without any Boolean operator is a condition [8].

3. Modified Decision / Condition Coverage: Modified Condition/Decision Coverage (MC/DC) [6] is a concept of ensuring adequate testing for safety-critical software. At its core lies the concept that if a choice can be made, all the possible factors (clauses) which contribute to that choice (predicate) must be tested [2].

4. CONCOLIC Testing: The CONCOLIC testing concept combines a concrete constraints execution and symbolic constraints execution to automatically generate test cases for full path coverage [14].

5. Cyclomatic Complexity: It is a software metric. The constraint calculated for Cyclomatic Complexity [5] shows the number of independent paths in the basis set of a C-program. Independent path is any path through a program that introduces at least one new set of processing expressions. For, the given graph G, cyclomatic complexity V(G)=E-N+2, where E is the
number of edges, and $N$ is the number of nodes or vertices. $V(G)=p+1$, where $p$ is the number of predicate [18] nodes.

3. RELATED WORKS

Godbole et al. [13] proposed an architecture to increase the Modified Condition / Decision Coverage using Program Code Transformer. Program Code Transformer is based on Quine-McCluskey minimization method. The objective of this paper is to automatically generate MC/DC [3] test suite. In our paper we have proposed concepts to calculate Cyclomatic Complexity and MC/DC Coverage percentage to analyse structured C programs.

Godbole et al. [14] proposed another approach to enhance the modified Condition/ Decision Coverage using exclusive- nor code transformer. The approach reduces the effort of minimizing the sum of product by simple X-NOR operator. This approach overcomes the disadvantages of old concepts. In our work, we have compared MC/DC Coverage percentage with Cyclomatic Complexity.

Godbole et al. [19] proposed a concept to measure Coverage Percentage by designing Code Slicer and Coverage Analysers, to introduce Program Slicing in Concolic Testing, so that researcher can achieve more coverage percentage.

In our paper we have calculated Coverage Percentage without any Godbole et al. [20] proposed concept for time constraints to calculate Coverage Percentage by designing Advanced Program Code Transformer and Coverage Analysers. In our paper we are not evaluating time constraints.

Godbole et al. [21] proposed an approach to generate test suite by using CREST tool for simulink/stateflow models generated from MATLAB. In our paper we are not working for any model, we are working for structured C programs.

Das et al. [15] proposed augmentation of MC/DC test case generation [9]. The approach deals with automatic generation of MC/DC test suite. The author proposed the concept by presenting Boolean Code Transformer (BCT). In our paper, We have worked without any transformation technique.

Bokil et al. [4] have proposed a tool AutoGen that reduces the costs and efforts for test data preparation by automatically generating test data for C code. AutoGen takes the C code and a criterion such as Statement Coverage, Decision Coverage, or Modified Condition/Decision Coverage (MC/DC) as input and generates non-redundant test data that satisfies the specified criterion. Bokil et al. [4] concluded that the effort required using the AutoGen tool was one third of the manual effort required. In our work, we have shown the cyclomatic complexity.

4. Proposed Concept

Our proposed concept is based on two parameters: Cyclomatic Complexity and MC/DC coverage percentage. In this section we have discussed regarding Cyclomatic Complexity by designing Complexity Calculator. We calculate MC/DC coverage percentage by using CREST tool and designing

4.1 Complexity Calculator

We can observe from Fig 1. that Complexity Calculator accepts structured C-Program to produce Cyclomatic Complexity. Complexity Calculator contains two steps: Predicate Identifier and Cyclomatic Complexity as shown in Algorithm 1 and Algorithm 2 respectively.

Algorithm1: Predicate Identifier.

Input: X // X is the program in C syntax
Output: P // P is the total number of predicate identified

Begin
for each statement $\in X$ do
if && or || occurs in s then
  1) List Predicate ← adding_in_List (s) // List of predicates
end if
end for
2) return P
End

Algorithm2: Cyclomatic Complexity.

Input: P // P is total number of predicates for X program
Output: V(X) // Cyclomatic Complexity

Begin
1) $V(X) \leftarrow P + 1$ // Cyclomatic Complexity Technique
2) return V(X)
End
The Algorithm 1 is to identify the predicates in C code under test. The calculator scans the C program character by character. Where ever the calculator detects Boolean operator “& &” or “| |” or “!” , the prototype selects whole line and saves in one .txt file. That file contains nothing but identified predicates in whole C Program. Here we are interested in number of predicates. Algorithm 1 return P i.e number of predicates.

Now, Algorithm 2 accepts the output of Algorithm 1 i.e P to calculate complexity. We all know that Cyclomatic Complexity is nothing but Number of predicates + 1, i.e V(X)=P+1. Algorithm 2 returns cyclomatic complexity.

4.2 MC/DC Coverage Percentage

The C-Program under test is passed to the CONCOLIC Tester i.e CREST TOOL [1]. This tester achieves branch coverage through random test generation. CONCOLIC tester is a combination of concrete and symbolic testing. The generated test cases depend on the path on each run. All test cases are stored in text files which form a test suite.

The Coverage Analyser determines the Coverage Percentage achieved by the test cases. We need to calculate the extent to which a program feature has been performed by the test suite. It also finds inadequacy of test cases and provides an insight on those aspects of an implementation that have not been tested. The entered program to be tested and the test data generated are passed to the Coverage Analyzer. Coverage Analyser (CA) calculates the extent to which the independent effect of the component conditions on the calculation of each predicate of the test data taken place. The MC/DC coverage achieved by the test cases T for the following program input X is calculated by the formula:

\[ \text{MC/DC coverage} = \left( \frac{\sum_{i=1}^{n} 1_{i} + \sum_{i=1}^{n} c_{i}}{100} \right) \times 100 \]  

(2)

Algorithm3: MC/DC COVERAGE ANALYSER

Input: X, Test Suite // Program X and Test Suite obtained

Output: MC/DC coverage // % MC/DC achieved for X

Begin
/* Identification of predicates */
for each statement sEX do
if & & or I occurs in s then
1) List_Predicate ← adding_in_List(s)
end if
end for

/* Determine the outcomes */
for each predicate p∈List Predicate do
for each condition c∈p do
for each test case td ∈ Test Suite do
if c evaluates to TRUE and calculate the outcome of p with td
then
2) True Flag ← TRUE
end if
if c evaluates to FALSE and calculate the outcome of p without td then
3) False Flag ← TRUE
end if

end for
if both True Flag and False Flag are TRUE then
4) I_List ← adding_in_List(c)
end if
5) C_List ← adding_in_List(c)
end for

/* Calculating the MC/DC coverage percentage */
6) MC DC COVERAGE ← (sizeof(I_List) sizeof(C List)) × 100%

5. ANALYSIS OF STRUCTURED C-PROGRAMS

In our experimental study, we have taken 17 complex programs. Table 1 shows the characteristics of 17 C-Programs with their lines of code (LOC), functions, predicates, and branches. Some programs are student assignments and some are open source programs. Phonex, ProgramSTE, ProgramTR, Sed, and Grep programs are very complex in nature, Since these are more than 1000 lines of code. Table 2 shows the MC/DC Coverage Percentage and Cyclomatic Complexity for all the 17 programs. Program named column shows the name of programs, M_Cov represents the MC/DC Coverage Percentage, and CC represent Cyclomatic Complexity. Four programs Triangle, Next Date, ATM, and Library achieve high coverage percentage more than 70% among all complex programs.

In Table 2 we can observe that five Complex programs Phonex, ProgramSTE, ProgramTR, Sed, and Grep having high Cyclomatic Complexity equal to or greater than 20. So here we may conclude the point that MC/DC Coverage Percentage is inversely proportional to Cyclomatic Complexity. This is dynamic in nature since we can observe programs Replace and Ptok1, here M cov for Replace is less than M cov of Ptok1 where as Ptok1 program having more Cyclomatic Complexity. So it varies program to program.

<table>
<thead>
<tr>
<th>Programs</th>
<th>Characteristics</th>
<th>MC/DC Coverage Percentage</th>
<th>CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProgramSTE</td>
<td>Student assignment</td>
<td>80%</td>
<td>30</td>
</tr>
<tr>
<td>ProgramTR</td>
<td>Student assignment</td>
<td>90%</td>
<td>40</td>
</tr>
<tr>
<td>Sed</td>
<td>Open source</td>
<td>85%</td>
<td>35</td>
</tr>
<tr>
<td>Grep</td>
<td>Open source</td>
<td>95%</td>
<td>50</td>
</tr>
<tr>
<td>Phonex</td>
<td>Open source</td>
<td>70%</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 1: Characteristics of C-Programs
Table 2: Analysis of Metrics

<table>
<thead>
<tr>
<th>S.No</th>
<th>Program</th>
<th>Loc</th>
<th>Functions</th>
<th>Predicates</th>
<th>Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Triangle</td>
<td>75</td>
<td>1</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>Next Date</td>
<td>135</td>
<td>6</td>
<td>3</td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>ATM</td>
<td>241</td>
<td>1</td>
<td>10</td>
<td>54</td>
</tr>
<tr>
<td>4</td>
<td>Library</td>
<td>242</td>
<td>6</td>
<td>4</td>
<td>66</td>
</tr>
<tr>
<td>5</td>
<td>TCAS</td>
<td>338</td>
<td>10</td>
<td>10</td>
<td>88</td>
</tr>
<tr>
<td>6</td>
<td>Schedule</td>
<td>349</td>
<td>16</td>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>7</td>
<td>Tic-tac-toe</td>
<td>375</td>
<td>6</td>
<td>11</td>
<td>126</td>
</tr>
<tr>
<td>8</td>
<td>Elevator</td>
<td>530</td>
<td>6</td>
<td>8</td>
<td>158</td>
</tr>
<tr>
<td>9</td>
<td>Tokenizer</td>
<td>578</td>
<td>19</td>
<td>6</td>
<td>162</td>
</tr>
<tr>
<td>10</td>
<td>Ptok2</td>
<td>672</td>
<td>24</td>
<td>9</td>
<td>168</td>
</tr>
<tr>
<td>11</td>
<td>Replace</td>
<td>733</td>
<td>20</td>
<td>15</td>
<td>200</td>
</tr>
<tr>
<td>12</td>
<td>Ptok1</td>
<td>893</td>
<td>19</td>
<td>18</td>
<td>284</td>
</tr>
<tr>
<td>13</td>
<td>Phonex</td>
<td>1198</td>
<td>33</td>
<td>19</td>
<td>348</td>
</tr>
<tr>
<td>14</td>
<td>ProgramSTE</td>
<td>1254</td>
<td>28</td>
<td>25</td>
<td>384</td>
</tr>
<tr>
<td>15</td>
<td>ProgramTR</td>
<td>1311</td>
<td>42</td>
<td>23</td>
<td>362</td>
</tr>
<tr>
<td>16</td>
<td>Sed</td>
<td>10143</td>
<td>70</td>
<td>48</td>
<td>2690</td>
</tr>
<tr>
<td>17</td>
<td>Grep</td>
<td>13743</td>
<td>126</td>
<td>53</td>
<td>3768</td>
</tr>
</tbody>
</table>

6. CONCLUSION AND FUTURE WORK

In this paper, we have developed an approach for Cyclomatic Complexity Analysis and evaluating MC/DC coverage.
percentage for structured C programs. We have discussed about Complexity Calculator, CREST Tool, and Coverage Analyzer with their working and description. In experimental studies we have done experiments on 17 complex programs. In that we have calculated the cyclomatic complexity and MC/DC coverage percentage. Hence, we conclude that the average MC/DC coverage percentage is indirectly proportional to Cyclomatic Complexity. Future version of this research work may contain cost estimation concepts.

7. ACKNOWLEDGMENT

We are indebted to Prof. Rajib Mall, IIT Kharagpur, Mr. Avijit Das of Advanced Systems Laboratory, Defence R & D Organisation, Hyderabad and Mr. Prasad Bokil of TRDDC, Pune for their suggestions and guidance for this research work. We are thankful to Mr. Praveen Sen, Head of Computer Science and Engineering and Dr. Surendra Gole, Principal, Nagpur Institute of Technology, Nagpur for their kind supports.

8. REFERENCES

[17] Kim, M. , Kim, Y. and Choi, Y. Concolic testing of the multi sector relief operation for flash storage platform software. Under Consideration for publication in Formal Aspects of Computing. CS Dept. KAIST, Daejeon, South Korea and School of EECS, Kyungpook National University, Daegu, South Korea, 2011