An Automated Tool for MC/DC Test Data Generation

Ariful Haque, Irman Khalil, and Kamal Z. Zamli
Faculty of Computer Systems and Software Engineering
Universiti Malaysia Pahang
26300 Kuantan, Pahang, Malaysia
arifulhb@gmail.com

Abstract— **Structural testing is often the most common sought criteria for exercising aspects of control flow (i.e. such as statement, branch and path coverage). In many cases, criteria based on statement, decision and path coverage appears sufficiently effective for testing (in terms of selecting the appropriate test cases for testing consideration) the various parts of the software implementation. In other cases involving complex predicates, criteria based on statement, branch, and path coverage appear problematic owing to the problem of masking (where one variable is "masking" the effects of other variables). Addressing this issue, this paper discusses the design and implementation of an automatic test data generation called MC/DC GEN for structural testing based on Multiple Condition/Decision Coverage (MC/DC). In doing so, this paper also highlights the possible adoption of MC/DC GEN for practical use.**

Index Terms – Coverage Testing, MC/DC Test Generation, Structural Testing

I. INTRODUCTION

Manual software testing is time-consuming and also uses a lot of resources. The National Institute of Standards and Technology (NIST) reported that the monetary loss owing to inadequate infrastructure for software testing was estimated from $22.2 to $59.5 billion [7] in 2022. In terms of labor, based on data from 2010 to 2011, the cost of preventable software bugs increased to at least $25.8 billion. Given this huge monetary implications, the development of automated tools for software testing is deemed inevitable.

Owing to the rapid growth of the software line of codes due to increasing demands for new functionalities, test engineers often under increasing pressure to select the best strategies and their implementation supports for testing consideration particularly in terms of test cases effectiveness as well as its associated costs [1]. Concerning structural testing, criteria based on statement; branch and path coverage; has been the most common [2][3]. In many cases, criteria based on statement, decision and path coverage is sufficiently effective for testing the various parts of the software implementation (in terms of selecting the appropriate test cases for testing consideration). In other cases involving complex predicates, criteria based on statement, branch, and path coverage appear problematic owing to the problem of masking (where one variable is "masking" the effects of other variables). To illustrate further, assume two basic predicates – (A or B) and (A and B) respectively. The predicate (A or B) always evaluates to true when either A is true (regardless of B) and vice versa. Similarly, the predicate (A and B) is always false when B is false (regardless of A) and vice versa. In this case, A and B are said to have masked each other. Addressing this issue, this paper discusses the design and implementation of an automatic test data generation, called MC/DC GEN, for structural testing based on Multiple Condition/Decision Coverage (MC/DC). Apart from the lack of automated tools to support MC/DC in the literature, the rational for adopting MC/DC stemmed from the fact that it is the main structural testing criteria required for mission critical software system as required by the DO-178B and DO-178C standards [9] (i.e. as part of Airborne Systems and Equipment Certification).

The rest of the paper is organized as follows. Section II describes the related work on MC/DC Section III gives a synopsis of the MC/DC. Section IV describes our tool design and implementation and finally, Section V gives our conclusion and scope for future work.

II. RELATED WORK

Ghani and Clark [2] introduce an automatic framework to extend the capabilities of search based testing technique for MC/DC. The advantage of this framework is that it can be used to test stronger coverage criteria such as Multiple Condition Coverage and MC/DC. Simulated annealing optimization technique is used into this framework. The framework is compared with other tools for software testing; Triangle, CalDate, Quadratic, Complex and Expint. The result shows that the search by this framework manages to get 100 % coverage for all tools except for Expint.

Jones and Harrold [4] have proposed an algorithm for MC/DC with prioritization. There are four steps involved to reduce the number of test cases. The steps are removing uncovered pairs, identifying test cases, assigning test cases contributions and removing weakest test cases. Here, prioritization involves two steps which are selecting the highest entity coverage and choosing the highest contribution values test cases.

Similar to Ghani and Clark[2], Awedikian et al. [1] adopts search based algorithm for generating MC/DC test suite. Unlike Ghani and Clark [2] which adopts Simulated Annealing, Awedikian et al adopts the Hill Climbing (HC) and Genetic Algorithm (GA) respectively.

In other work, Jun-Ru and Chin-Yu [5] usefully exploit n-cube graph in order to generate appropriate MC/DC compliant test data. In this case, the vertex of the cube represents the resultant Boolean enumeration for predicates under evaluation. Each vertex is traversed and arranged and evaluated using Gray code sequence ordering until all the required sequences are covered. A tool called TASTE (Tool for Automatic Software Regression Testing) has been developed as a result.

Kandl and Kirner [6] exploit MC/DC to automotive domain. The goal of their study is to inspect the error detection rate of a set of test that attains maximum possible MC/DC coverage. The first stage was done by generating the test cases. Here, test cases are generated using a model checker followed by transforming the program into three different errors circumstances. The results proved that fewer errors were detected when a system was tested with a set of test that attains maximum possible MC/DC on the code.

III. Overview of Modified Condition/Decision Coverage

As highlighted earlier, the Modified Condition/Decision Coverage is a structural testing coverage criteria advocated by NASA for testing safety critical software applications. Exists in pairs, MC/DC criteria [11] insists that each variable at the atomic level is able to independently influence the overall outcome – while keeping other variable(s) unchanged. Specifically, each one of the pair differs only by the Boolean value of one condition, but gives a different result for the decision statement. For AND operation, MC/DC pairs are {{F,T}, {T,T}}, {{T,F}, {T,T}}. As the entry {T,T} is redundant, the complete MC/DC compliant test predicate is reduced to {F,T}, {T,F} and {T,T}. In similar manner, for OR operation, the MC/DC pairs are compliant test predicates are {{F,F}, {T,F}}, {{F,T}, {T,F}}.

As a running example, consider the following if statements involving AND, OR, and NOT operations (see Fig. 1). To facilitate discussion, we have assigned the predicate (x>100) = A, (y<50) =B, and (Z) = C. Here, as long as A&&B holds TRUE, statement 1 will always be executed regardless of the value of C. Similarly, given that NOT C is TRUE, statement 2 will always get executed regardless of the values of A&&B. In this manner, the resulting predicate is masking each other given the wrong selection of inputs values. Referring from the given truth table in Fig. 1, there are:

- 1 pair for MC/DC coverage for variable A
- 1 pair for MC/DC coverage for variable B
- 3 possible pairs for MC/DC coverage for variable C

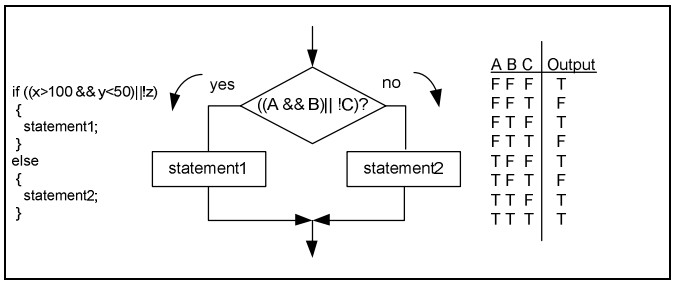

Fig. 1. Illustrative Example

These pairs are summarized as follows:

A = {F T T| F}, {T TT| T}
B = {T F T | F}, {T TT| T}
C = {F FF| T}, {F FT | F}
C = {F T F| T}, {F T T| F}
C = {T F F| T}, {T FT | F}

Combining the pairs together and removing repetition yield three possible MC/DC solutions:

$$\text{Result 1} = \left\{ \begin{array}{c} F\ T\ T \mid F \\ T\ F\ T \mid F \\ T\ T\ T \mid T \\ F\ F\ F \mid T \\ F\ F\ T \mid F \end{array} \right\} \quad \text{Result2} = \left\{ \begin{array}{c} T\ F\ T \mid F \\ T\ T\ T \mid T \\ F\ T\ F \mid T \\ F\ T\ T \mid F \end{array} \right\}$$

$$\text{Result3} = \left\{ \begin{array}{c} F\ T\ T \mid F \\ T\ T\ T \mid T \\ T\ F\ F \mid T \\ T\ F\ T \mid F \end{array} \right\}$$

Finally, any of the MC/DC results can be converted to the appropriate test cases satisfying the predicates. Taking the first result "Result 1" as example, the test cases can be generated as follows:

TABLE I. Possible MC/DC Compliant Test Suite

Test ID	x>100	y<50	!z	Expected Value
1	90	30	True	False (execution of statement 2)
2	110	60	True	False (execution of statement 2)
3	105	35	True	True (execution of statement 1)
4	80	70	False	True (execution of statement 1)
5	20	94	True	False (execution of statement 2)

Although it is possible to generate MC/DC compliant test manually, the generation process can become tedious when dealing with large number of predicates. Instead, much of these efforts can be put to use for other (testing) activities with the

automation support. The development of this automated tool, called MC/DC Gen, is the main focus of this work.

IV. DESIGN AND FRAMEWORK FOR MC/DC GEN

Before the implementation of MC/DC GEN is further discussed, the design of this application needs to be elaborated accordingly. We have opted to develop MC/DC GEN using the PHP programming language and hosted in a Linux based Virtual Private Server. The main reason for such a choice stemmed from the fact that we want our tool to be accessible through the web and available 24/7.

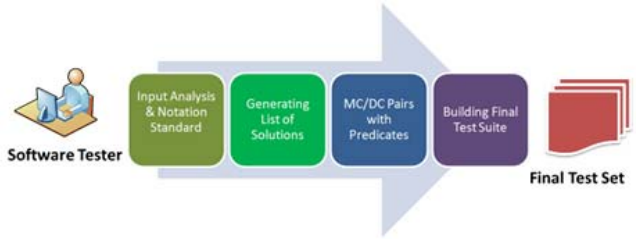

Fig. 2. MC/DC GEN Components

Figure 2 depicts the overall components for MC/DC GEN where each of these components is discussed in the next paragraphs.

A. *Input Processing and Analysis*

To initiate the generation process, test engineers need to input the predicates (Boolean expression) according to the MC/DC Gen application standard. For AND operation, user can use "." or "&" notation. For OR operator, user can user "+" or "|" notation. "!" should be use to for negation (NOT operator). For grouping parameters "()" should be used. A sample expression is given in Figure 3.

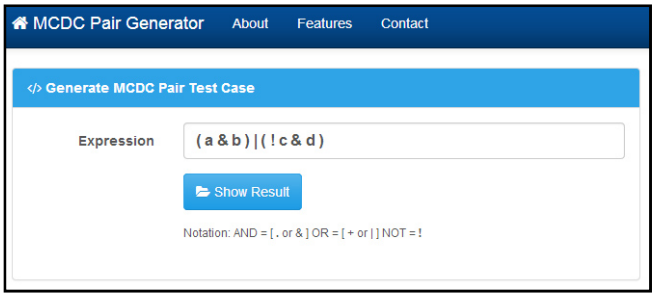

Fig. 3. Input Processing

Upon input submission, internally MC/DC Gen separates the predicates (e.g. a, b, c, d), operators (e.g &, |, !) and grouping notation from the expression whilst keeping a pointer to match the right predicate with the right operator.

B. *Generating List of Solutions*

MC/DC Gen starts with a local search (loop) to find all possible solutions of separated predicates. Before starting the search, a process randomly choose an initial solution 'i' and compute its result. An item is a row of truth table for separating predicates. During the search, MC/DC Gen seeks for a neighbor solution 'n' of 'i', and computes its result n(r). A neighbor is identified by changing value of predicate. If i(r) != n(r) that's mean n(r) is pair of i(r) and its stored as a MC/DC Pair array '$pairs[]' variable. After that, the loop iterates and identifies the next neighbor as the candidate solution. The candidate solution is kept as variable to store its predicate information pair.

If result of i(r) != n(r) becomes false, the loop then iterates and chooses a random solution along with its possible neighbor. This process continues until all the predicates are searched completely. A log of how random items and their neighbor items are listed (i.e. including MC/DC pair) is presented in Figure 4.

SN	SP	Current	(a&b)|(!c&d)	Neighbour	N-Result	Pair
0	0	0000	0	1000	0	
1	1	0000	0	0100	0	
2	2	0000	0	0010	0	
3	3	0000	0	0001	1	Array ([Test Case] => 0000 [Pair] => 0001)
4	0	0001	1	1001	1	
5	1	0001	1	0101	1	
6	2	0001	1	0011	0	Array ([Test Case] => 0001 [Pair] => 0011)
7	3	0001	1	0000	0	Array ([Test Case] => 0001 [Pair] => 0000)
8	0	1001	1	0001	1	
9	1	1001	1	1101	1	
10	2	1001	1	1011	0	Array ([Test Case] => 1001 [Pair] => 1011)
11	3	1001	1	1000	0	Array ([Test Case] => 1001 [Pair] => 1000)

Fig. 4. Log of Items and Neighbor Items with Result

C. *MC/DC Pairs with Predicates*

From the data stored in an internal array, MC/DC Gen is able to populate a table with predicates in column and list of identified MC/DC pairs for each predicate in row. This representation is an overall view of identified solutions (see Figure 5).

MCDC Pairs by Predicates

For [a]	For [b]	For [c]	For [d]
TC: 0100, Pair: 1100 TC: 0111, Pair: 1111 TC: 0110, Pair: 1110	TC: 1100, Pair: 1000 TC: 1110, Pair: 1010 TC: 1111, Pair: 1011	TC: 0001, Pair: 0011 TC: 0101, Pair: 0111 TC: 1001, Pair: 1011	TC: 0000, Pair: 0001 TC: 0101, Pair: 0100 TC: 1000, Pair: 1001

Fig. 5. Identified MC/DC Pairs by Predicates

D. Generating the Final Test Data

The pairs of each predicates are identified in the previous step. In order to generate the final test data, the duplication test data in previous step need to be removed. From the pairs, there are duplicate test data for parameter B and parameter C. The second test data from parameter B {0100} is also used by parameter C. This duplication will be removed and only one pair will be maintained. In this case, the second test data of parameter C will be removed.

MC/DC Gen performs this task through a unique() array construct provided by PHP. After that, MC/DC Gen generates all the actual MC/DC pairs from the combination of pairs. The final results are shown in Figure 6.

MC/DC Pairs for (a&b) | (!c&d)

MC/DC Pairs Set 1: 0100, 1100, 1000, 0001, 0011, 0000
MC/DC Pairs Set 2: 0111, 1111, 1100, 1000, 0001, 0011, 0000
MC/DC Pairs Set 3: 0110, 1110, 1100, 1000, 0001, 0011, 0000

Fig. 6. MC/DC Gen Result

E. Case Study: MC/DC vs Pairwise

To benchmark the effectiveness of MC/DC for structural testing, the competing pairwise [10] technique has been selected. In a nut shell, the pairwise technique is a testing strategy based to sample all possible discrete combination of involved parameters with interaction strength of two. The rationale for adopting the pairwise technique is based on the observation that many real-life faults are caused by two way interaction between parameters.

Here, a case study consisting of a partial "IF" statement is adopted for evaluation purposes (see Figure 7).

```
2
3   if (($a==10)||($b<100 && $c>90 && $d!=15)){
4       echo "Switch on";
5   }else{
6       echo "Switch off";
7   }
8
```

Fig. 7. IF Statement with Four Variables

In order to get the value of each variable (or parameter) using pairwise, Boundary Values Analysis (BVA) technique was applied. Using boundary value analysis, the value of "TRUE" and "FALSE" from each condition can be derived. From the "IF" statement, the value of each parameter can be described as in Table II.

TABLE II. PAIRWISE VALUES

PARAMETERS	VALUES	
A	{1a,1b}	{8, 10}
B	{2a, 2b}	{50, 110}
C	{3a, 3b}	{80, 100}
D	{4a, 4b}	{5, 15}

In this case, the total parameter is four and each parameter has two values. To generate minimum possible combination for pairwise technique, a tool called Jenny is used. Jenny is a free open source pairwise test generation program written in C. To get the possible pairwise combination, we will use the information in Table 3, the interaction strength as t=2. The complete output is presented in Figure 8:

```
C:\>jenny -n2 2 2 2 2
 1a 2b 3b 4b
 1b 2a 3a 4a
 1a 2a 3a 4b
 1b 2b 3b 4a
 1a 2b 3a 4a
 1b 2a 3b 4b
```

Fig. 8. Using Jenny to generate Pairwise test data

Using the value from Table 3 and based on the Jenny's output, the test cases are prepared accordingly. In Jenny, it should be noted that each output parameter is identified with a number along with a corresponding alphabet. Here, the number represents the parameter/column whilst the alphabet represents corresponding value. For instance, 1a represents the value of 8, and 4b represents the value of 15.

Using the generated results from Jenny, the following test cases have been generated as depicted in Table 3.

TABLE III. PAIRWISE TEST CASES

TEST CASE	VALUES				EXPECTED OUTPUT	REAL OUTPUT
	a	b	c	d		
1	10	50	100	15	Switch On	Switch On
2	8	110	80	5	Switch Off	Switch Off
3	10	50	80	5	Switch Off	Switch Off
4	8	110	100	15	Switch Off	Switch Off
5	8	50	100	5	Switch On	Switch On
6	10	110	80	15	Switch Off	Switch Off

In order to generate the test cases for MC/DC, the Boolean expression of "($a==10)||($b<100 && $c>90 && $d!=15)" need to be abstracted first. Let A="$a==10", B= "$b<100", C="$c>90", and D="$d!=15", then the expression "A|(B&C&D)" represents the overall Boolean expression. The expression is fed into MC/DC gen to produce 8 MC/DC pairs. The pairs for each parameter are described as:

A = {0000|0} {1000|1}, B = {0011|0} {0111|1},
C = {0101|0} {~~0111|1~~}, D = {0110|0} {~~0111|1~~}

Removing the repetition, the output for MC/DC Gen is:

{0011|0}, {1000|1}, {0101|0}, {0110|0}, {0000|0}, {0111|1}

In order to proceed, there is a need to interpret the output from MC/DC Gen. As illustration, consider the value of {0011|0}. In this case, {0011|0} represents A=false, B=false, C=true, D=true with the output of false. Here, the corresponding value for variable A="$a==10" must meet the condition to be false. The same goes for other variables also.

Table 4 maps the possible values for each parameter.

TABLE IV. MC/DC TEST CASES FOR "A|(B&C&D)"

TEST CASE	VALUES				EXPECTED OUTPUT	REAL OUTPUT
	a	b	c	d		
1	50	200	150	10	Switch Off	Switch Off
2	10	120	15	15	Switch On	Switch On
3	40	50	8	5	Switch Off	Switch Off
4	80	10	115	15	Switch Off	Switch Off
5	50	150	5	15	Switch Off	Switch Off
6	30	20	110	8	Switch On	Switch On

In order to compare between Pairwise and MC/DC, the selected program is decomposed into similar program with added marker for every branch to test the branch coverage.

There are five branches in the given case study. Figure 9 depicts the available branches.

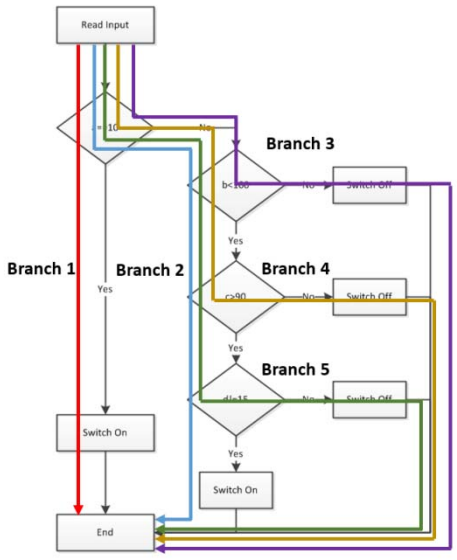

Fig. 9. Branches in the Case Study Program

Test cases from pairwise technique are used to check the branch coverage. Table 5 depicts the results from the tests. Referring to Table 5, the number of possible combination was reduced from 16 to 6. For the branch coverage, Pairwise technique has achieved only 3 branches from total of 5. In this test, the Pairwise technique was unable to cover Branch 4 and Branch 5. This is about 60 % of branch coverage that is successfully covered by the Pairwise technique.

For the results of MC/DC coverage, the total possible combination was also reduced from 16 to 6. For the results of branch coverage in Table 6, MC/DC manages to cover 5 from the total of 5 branch coverage (i.e. 100% coverage).

TABLE V. PAIRWISE BRANCH COVERAGE

TEST CASE	VALUES				REAL OUTPUT	STATUS	BRANCH COVERAGE
	a	b	c	d			
1	10	50	100	15	Switch On	Passed	Branch 1
2	8	110	80	5	Switch Off	Passed	Branch 3
3	10	50	80	5	Switch Off	Passed	Branch 1
4	8	110	100	15	Switch Off	Passed	Branch 3
5	8	50	100	5	Switch On	Passed	Branch 2
6	10	110	80	15	Switch Off	Passed	Branch 1

TABLE VI. MC/DC BRANCH COVERAGE

TEST CASE	VALUES				REAL OUTPUT	MC/DC Results	STATUS	BRANCH COVERAGE
	a	b	c	d				
1	50	200	150	10	Switch Off	0011|0	Passed	Branch 3
2	10	120	15	15	Switch On	1000|1	Passed	Branch 1
3	40	50	8	5	Switch Off	0101|0	Passed	Branch 4
4	80	10	115	15	Switch Off	0110|0	Passed	Branch 5
5	50	150	5	15	Switch Off	0000|0	Passed	Branch 3
6	30	20	110	8	Switch On	0111|1	Passed	Branch 2

V. CONCLUSION

Summing up, this paper has highlighted the development of MC/DC Gen along with its practical use for MC/DC test data generation. Comparative experiment with pairwise testing technique demonstrates the superiority of MC/DC for structural testing. As the scope of future work, we plan to automate the actual generation of test from the MC/DC generated pairs.

ACKNOWLEDGMENT

The authors would like to thank Universiti Malaysia Pahang for the financial support from the grant Graduate Research Scheme (UMP.20.01/13/13.14/1/26).

REFERENCES

[1] Awedikian, Z., Ayari, K., & Antoniol, G. "*MC/DC Automatic Test Input Data Generation*." in Proceeding of the 11th Genetic and Evolutionary Computation Conference, Montreal, Québec, Canada, pp-1657-1664, 2009.

[2] Ghani, K., & Clark, J. A. "*Automatic Test Data Generation for Multiple Condition and MC/DC Coverage*." in Proceeding of the 4th International Conference on Software Engineering Advances, Porto, Portugal, pp-152-157, 2009

[3] Hayhurst, K. J., Veerhusen, D. S., Chilenski, J. J., & Rierson, L. K. *A Practical Tutorial on Modified Condition/ Decision Coverage,* (Vol. NASA Technical Memorandum TM-2001-210876). 2001 NASA Langley Research Center.

[4] Jones, J A., and Mary Jean Harrold. "*Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage.*" IEEE Transactions on Software Engineering, Vol. 29, Issue 3, pp-195-209, 2003.

[5] Jun-Ru, C., & Chin-Yu, H. 2007. "*A Study of Enhanced MC/DC Coverage Criterion for Software Testing."* In Proceeding of 31st Annual IEEE International Computer Software and Applications Conference, Beijing, 2007, pp-457-464.

[6] Kandl, S., & Kirner, R. "*Error detection rate of MC/DC for a case study from the automotive domain*." in Proceeding of the 8th IFIP WG 10.2 International Workshop, Waidhofen/Ybbs, Austria, 2010, pp-131-142.

[7] Tassey, Gregory. "*The Economic Impacts of Inadequate Infrastructure for Software Testing.*" National Institute of Standards and Technology, RTI Project 7007, no. 011, 2002.

[8] *SQS Identifies the highest profile software failures of 2011* Retrieved 30 March 2013, Available in web: http://www.sqs.com/en/group/about-sqs/press-archive_6399.php

[9] Hayhurst, Kelly J., and Dan S. Veerhusen. "*A Practical Approach to Modified Condition/Decision Coverage.*" in Proceeding of 20th Conference of Digital Avionics Systems, Vol. 1, 2001, pp. 1B2-1.

[10] Younis, M.I., K. Z. Zamli, and N. A. M. Isa, "*IRPS -- An Efficient Test Data Generation Strategy for Pairwise Testing,*" in Proceedings of the 12th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, Part I, Zagreb, Croatia, 2008.

[11] Zamli, K. Z., AbdulRahman A. Al-Sewari, and Mohd Hafiz Mohd Hassin. "*On Test Case Generation Satisfying the MC/DC Criterion.*" International Journal of Advances in Soft Computing & Its Applications 5, no. 3, 2013.