Advances in Computer Science and Information Technology (ACSIT)
Print ISSN: 2393-9907; Online ISSN: 2393-9915; Volume 2, Number 8; April-June, 2015 pp. 37-40

http://www.krishisanskriti.org/acsit.html

Metamorphic Testing as a Testing Technique and an Automated Debugging Tool

Jasmine Kaur[1] and Harpreet Kaur[2]

[1]Department of Computer Engineering Punjabi University Patiala Punjab, India
[2]Faculty of Department of Computer Engineering Punjabi University Patiala Punjab, India
E-mail: [1]jasminekhurma@yahoo.com, [2]moonkhurma@gmail.com

Abstract—*Metamorphic Testing (MT) is an efficient methodology for testing those programs which are sometimes called as "Non-testable", this is because most of the times it becomes complicated for testers to know whether the imminent outputs are accurate or not. This is the case in such category of programs, when the test oracles don't exist. In MT, oracle is not the necessary condition and it depends only upon the Metamorphic Relations (MR). The pioneer researcher T.Y.Chen, has implemented this technique on diverse programs like Partial Differential Equations, Sine function, Median, Area, Replace, Wireless Embedded System Software, Group Theory, etc. Now a days, MT is integrated with some other techniques. The use of metamorphic testing technique firstly started with the testing of the numerical programs however, presently its coverage has successfully widened to test the non-numerical programs too. But metamorphic testing (MT) is dependent upon the properties of programs under test (PUT), from which metamorphic relations (MR) can be obtained. In this paper, MT has been applied on the two mathematical programs, to demonstrate its usefulness. This paper also examines the MR's and the extent of their usefulness in the program testing. . It has been observed that a strong metamorphic relation is a relation where it involves the execution of the interior functionality and as such would efficiently validate the specified function. Hence, the most critical type of error which is known as missing path error has been discovered in one of the program. So, by this way the effectiveness of metamorphic testing has been proved. The debugging of these programs has also been done. The method used in the paper also supports automatic debugging through the detection of constraint in expressions that divulge failures.*

1. INTRODUCTION

Software testing is an important step to complete the process of software development. Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. There are many techniques to test the software, with the help of these techniques, the process of executing a program or application with the purpose of finding software bugs (errors or other defects) is implemented. Software testing can be done as soon as executable software (even if partially complete) is ready to be completed. An oracle is used in software testing to check whether the program under test (PUT) produces the expected output when executed using a set of test cases. A program is considered non-testable [1] when an oracle does not exist or is impractical to implement even though theoretically possible. So, the testing technique called as Metamorphic Testing (MT) is the best solution to find out the errors and to debug those too, so in this kind of situation if any other testing technique is applied then tracking of errors is not definite. While, in MT oracle is not the necessary condition and it depends only upon the Metamorphic Relations (MR). The pioneer researcher T.Y.Chen, has implemented this technique on diverse programs like Partial Differential Equations, Sine function, Median, Area, Replace, Wireless Embedded System Software, Group Theory, etc. He has also integrated MT with Semi-Proving technique and with Program Slicing. He applied this technique earlier only on the programs which were numeric. Recently, T.Y.Chen as well as many other researchers have integrated MT with Fault-Based testing and have also practiced checkpoints for enhancement in MT technique.

In this paper, after introduction in section 1, meaning and procedure of metamorphic testing are discussed in section 2. Section 3 contains the implementation of MT on trisquare program and its results in tabular form. The testing of a Median program using MRs is done in section 4. The conclusion is presented in section 5.

2. METAMORPHIC TESTING

Metamorphic testing was proposed to overcome some of the inherent problems that are present in testing software without test oracles [2,3]. It is a software testing technique that attempts to alleviate the test oracle problem. A test oracle is the mechanism by which a tester can determine whether a program has failed. Metamorphic testing involves the verification of a metamorphic relation **R** of function **f**, of which **p** is an implementation. The concept of metamorphic relation **R** has a number of characteristics. Firstly, metamorphic relationships are necessary properties of the target function **f**. If any of these properties does not hold

during testing, the program **p** is faulty. Secondly, a metamorphic relation is a relationship among the inputs and outputs involving multiple executions of **p [3]**. These Metamorphic Relations offer a new viewpoint on verifying test results. Unlike traditional testing, Metamorphic Testing always involves multiple test case executions with their corresponding outputs being verified using the Metamorphic Relations rather than a test Oracle. The formation of MRs is not restricted to identity relations. Any expected relation linking inputs and outputs of two or more executions of the program can be taken as an MR. The basic idea of MT is like this: Given a series of inputs satisfying a certain condition, if another condition should hold in the corresponding series of outputs, then we can check the relation between the input and output conditions to verify the correctness of the program. The relation associating the conditions of the series of inputs and the series of outputs is known as a metamorphic relation. When a metamorphic relation breaks, we shall definitely know that the implementation under test contains fault(s). In other words, program ***p*** must appear in the relation more than once. Although special value testing provides a way to test program correctness, there is no theoretical ground why special value testing is sufficient in ensuring program correctness. Metamorphic testing complements special value testing by its ability to test the necessary properties of the function. Moreover, the test data set used by metamorphic testing is augmented by using random test values.

The examples used in this paper follow the general approach where a) a set of special values and expected test outputs are selected to be used in the testing; b) a fault is introduced to a test program, called a mutant program; c) metamorphic relations for the function are defined; d) perform metamorphic testing as well as special value testing.

3. METAMORPHIC TESTING OF TRISQUARE PROGRAM

Here we use the typical triangle program called Tri-Square to conduct metamorphic testing. Type and square of this triangle are calculated The seven metamorphic relations have been constructed in literature [6]. This program first decides whether 3 positive real numbers, a, b and c, could construct a triangle. The input domain of the program TriSquare can be divided into arbitrary triangle class, equilateral triangle class and isosceles triangle class which is a composite equivalence class. The isosceles triangle class can be further divided into a = b isosceles triangle class, a = c isosceles triangle class and b = c isosceles triangle class (a, b, and c are the different edges of the same triangle). We construct 7 binary MRs for TriSquare, details are shown in Table 1.

Table 1: Metamorphic Relations

MR	Dbl (mri)	Relations
MR1	(a', b', c') = (b, a, c)	For 4 MR TriSquare(a' ,b' ,c') =4×TriSquare(a,b,c) Others TriSquare(a' ,b' ,c') =TriSquare(a,b,c)
MR2	(a', b', c') = (a, c, b)	
MR3	(a', b', c') = (c, b, a)	
MR4	(a', b', c') = (2*a, 2*b, 2*c)	
MR5	(a' ,b' ,c')=(2b2+2c2 -a2 ,b,c)	
MR6	(a' ,b' ,c')=(a, 2a2+2c2 -b2 ,c)	
MR7	(a' ,b' ,c')=(a,b, 2a2+2b2 -c2)	
Note: The domain of metamorphic relations is {(a,b,c): (a+b>c) ⊥ (b+c>a) ^ (a+c>b))}		

3.1. Injecting the Faults

Mutation analysis is a powerful technique to assess the quality of a test suite [4]. As MRs' performance is embodied by MT test suite, we use mutation analysis here to estimate test suites and MRs. Four mutants are imported into *TriSquare* based on two types of mutant operators: AOR(Arithmetic Operator Replacement) and DSA(Data Statement Alterations) [5]:

Mutant1: Exchange sentence 2 and 3.

Mutant2: Replace sentence 11 with "p= (a+b+c)*2".

Mutant3: Replace "/2" in 18, 23 and 28 with "*2".

Mutant4: Replace sentence 30 with "return sqrt(3)*a*a/2".

3.2. Measurements for Test Suite and MR

The effectiveness of a testing method can be measured quantitatively at two levels in different granularities: one is by counting the number of mutants that could be detected; and the other by calculating how many of its test cases are able to detect a particular mutant.

MR Detection Performance (MDP):

Relation	Special test values	1,2,3				2,2,1				7,7,7				5,7,6			
		M1	M2	M3	M4	M1	M2	M3	M4	M1	M2	M3	M4	M1	M2	M3	M4
L.H.S	R.H.S	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
a,b,c	b,a,c																
a,b,c	a,c,b	T	F	F	T	T	T	T	T	T	T	T	T	T	T	T	T
a,b,c	c,b,a	T	T	T	T	F	T	F	T	T	T	T	T	T	T	F	T
a,b,c	2*a, 2*b, 2*c	T	T	T	T	F	F	F	F	F	F	F	F	F	F	F	F
a,b,c	Sqrt((2*b*b)+(2*c*c)-(a*a)),b,c	T	T	T	T	F	F	F	F	F	F	F	F	T	F	F	T
a,b,c	a,Sqrt((2*a*a)+(2*c*c)-(b*b)),c	T	F	F	T	F	F	F	F	F	F	F	F	T	F	F	T

a,b,c	a,b,Sqrt((2*a*a) + (2*b*b)-(c*c))	F	F	F	T	T	T	T	T	F	F	F	F	T	F	F	T

Measure the mutation detection performance of mr in terms of test suite TC, which is the number of mutants that the test cases using mr could find:

$$MDP(mr,TC) = \sum_{i=1}^{n} FindMut_i \quad (5)$$

FindMuti is 1 if at least one test case that uses mr could detect the ith mutant, otherwise is 0, and n is the number of mutants.

MR Detection ratio for each Mutant (MDM):

Measure the detection performance of MR mr for each mutant in terms of test suite TC, which is thepercentage of failed test cases using mr : MDM(mr,mp,TC) = Nmrf Nmrmp (6) , where mp is a mutant program, Nmrf the number of test cases which use mr and could detect the mutant in mp, and Nmrmp the number of test cases that use mr and executes paths including mutants.

We have implemented MT on trisquare program and calculated MDM and MDP values for it.

Table 2: Result of MDM and MDP value for Trisquare program

MR	MDM1	MDM2	MDM3	MDM4	MDP
mr1	0%	0%	0%	0%	0
mr2	30%	40%	30%	0%	3
mr3	10%	0%	40%	0%	2
mr4	60%	60%	60%	60%	4
mr5	30%	60%	60%	30%	4
mr6	50%	100%	100%	40%	4
mr7	60%	90%	90%	30%	4

Table 2 shows the MDM and MDP values of program Trisquare. It has been shown that mr6 is the powerful relation which have highest mutant detection ratios and mr1 is the weakest of all relations.

4. METAMORPHIC TESTING OF MEDIAN PROGRAM

Fig. 1 shows a program *Med*. It accepts three integers u, v and w as input and returns their median as output. The program is adapted from, where it was used as a worst-case example to illustrate the technique of constraint-based test case generation for mutation testing. Here three integers are given as input, and according to their corresponding values output comes. And hence depending upon different conditions seven metamorphic relations have been developed.

4.1. Injection of faults in the program

The two type of faults were injected in the program Med to perform the task of metamorphic testing. The first fault (Fault 1) was created by deleting the 5 and 6 line from the code. And then second fault (Fault 2) was generated by removing 9 and 10 lines from the same code. These two faults have been divided into two zones i.e Zone 1 and Zone 2, for the purpose of debugging. The faulty programs are named as medbar and medbar1. These faults are known as missing path errors which are very difficult to find out by any other simple test cases. Hence MT has been used to check its efficiency against them.

Table 5: Table showing Faults of program medianbar

Faults	Deleted paths	Zones
1	If (U < W) Then Med = U	Zone 1
2	If (U > W) Then Med = U	Zone 2

```
int Med (int u, int v, int w) {
int med;
1  med = w;
2  if (v < w)
3     if (u < v)
4      med = v;
      else {
5     if (u < w)
6     med = u;  }
  else
7     if (u > v)
8      med = v;
      else {
9     if (u > w)
10     med = u;  }
11 return med; }
```

Fig. 1: Program Med

```
int Medbar (int u, int v, int
w){
int med;
1  med = w;
2  if (v < w)
3     if (u < v)
4      med = v;
      else {


  else
7     if (u > v)
8      med = v;
      else {
9     if (u > w)
10     med = u;  }
11 return med; }
```

Fig. 2: Program Medbar

The programs are shown above in Fig. 1 and Fig.2. In which Med is the correct program and Medbar is the program which contain errors in it. The different test values of u, v and w depending upon the eight MRs have been used as input values. Then the expected value of Medbar was compared with actual value which is coming after the execution. The results are shown in Table 6.

4.2. Results analysis of Medbar program

We have found that for the test values 2,3,1 the expected and actual outputs are not same. As for fp(x) which is expected value, it is 2 and 1 for fp'(x') after execution of the faulty program. But it has been detected that where the error exist with the help of metamorphic relations. As here the error exists in MR2, which lies in Zone2. And we have described before in Table 5 that Zone 2 is the error zone for the missing path error.

So, here the missing path which has been detected is "If (U > W) Then Med = U".

Table 6: Result from metamorphic testing of medbar program fp'(x')

Special Test Values	Metamorphic Relation's		fp(x)= expected result	fp'(x')= actual result	Zone
1,2,3	MR1	U<W>V	2	2	___
2,3,1	MR2	U>W<V	2	1	Zone 2
3,2,3	MR3	U=W>V	3	3	___
1,2,1	MR4	U=W<V	1	1	___
2,3,3	MR5	U<W=V	3	3	___
1,3,2	MR6	U<W<V	2	2	___
3,1,2	MR7	U>W>V	2	2	___
2,1,3	MR1	U<W>V	2	2	___
3,2,1	MR2	U>W<V	2	2	___

5. CONCLUSION

This paper demonstrates the effectiveness of metamorphic testing. Metamorphic testing can reveal faults effectively as compared to any other special value testing technique. Both example programs used in this paper lack test oracles. Metamorphic testing alleviates this issue through testing of metamorphic relationships. The most critical type of error which is known as missing path error has been detected in one of the program. Hence, proving the effectiveness of metamorphic testing. It has been concluded that a strong metamorphic relationship is a relationship where it involves the execution of the core functionality and as such would effectively verify the function. A strong metamorphic relationship also has a high sensitivity to fault meaning that the relationship does not hold true for most input data. This is due to the nature of the faults and how they are related to the metamorphic relation. Hence, metamorphic testing could provide an effective way to detect errors in programs where test oracle is lacking and hence with the help of metamorphic relations, the automatic debugging process can also be done. But like other testing approaches, metamorphic testing only demonstrates the presence but not the absence of faults. So In other words, metamorphic testing does not prove the correctness of the program and so it should be used by integrating it by any other testing technique in future to increase its effectiveness.

REFERENCES

[1] Upulee Kanewala and James M. Biema, "Techniques for Testing Scientific Programs without an Oracle", IEEE Computer Society, 2013, pp 48-57.

[2] T.Y. Chen, T.H. Tse and Z. Q. Zhou, "Semi-proving: an integrated method based on global symbolic evaluation and metamorphic testing", In the Proceedings of the International Symposium on Software Testing and Analysis, 2002, pp.191-195.

[3] T.Y. Chen, T. H. Tse and Z.Q. Zhou, "Fault-based testing without the need of oracles", Information and Software Technology, vol.45, no. 1, 2003, pp.1-9.

[4] P. Wu, X.C. Shi, J.J. Tang and T.Y. Chen, "Metamorphic Testing and Special Case Testing: A Case Study", Journal of Software, 2005. pp. 1210-1220.

[5] A.J. Offutt, G. Rothermel, and C. Zapf, "An Experimental Determination of Sufficient Mutant Operators", ACM Transactions on Software Engineering and Methodology, 1996, pp. 99-118.

[6] Guowei Dong, Changhai Nie and Lulu Wang, "An Effective Iterative Metamorphic Testing Algorithm Based on Program Path Analysis", in Seventh International Conference on Quality Software, IEEE Computer society, 2007.