Metamorphic Testing Effectiveness on Trignometry

[1]Tarun Khosla, [2]Sushil Garg
[1,2]RIMT-IET, Mandi Gobindgarh, Punjab, India

Abstract
A test oracle in software testing is a mechanism for verifying whether the component under test behaves correctly for any execution. In some cases the oracles are unavailable or too difficult to apply. Fault based testing is used to fig. out the test set problem. In this paper we present the basic concept of metamorphic testing and extensible fact of testing any program.

Keywords
Metamorphic testing, metamorphic relation, test cases.

I. Introduction

It is a non practical task, to test the program with all conceivable inputs. The task and computations to find the faults are very much expensive. Instead we should focus on those test cases which reveal faults in the program. One of major limitation of software testing is oracle problem. An oracle is a mechanism against which we can decide whether the outcome of the program is correct or not. In certain cases it is very difficult or impractical to verify the correctness of the output of the program [1, 2]. On the other hand, when the oracle is available, if it is a human tester, the manual predictions and comparisons of the test results are often time consuming and error prone. Fault based testing on successful execution of program indicates the absence of faults. But this is not the true factor always.

At this stage, metamorphic testing can be carried out to generate follow-up test cases based on existing test cases that have not revealed any failure. A metamorphic relation (MR) is an expected relation among the inputs and outputs of multiple executions of the target program. In this paper we present the basic concept of metamorphic testing and how it effects the results testing of trigonometric component.

II. Metamorphic Testing

Metamorphic testing is proposed to alleviate the oracle problem. Its concept is simple and its automation is easy. Metamorphic testing is a program testing technique that employs the mathematical relations, namely metamorphic relations, to conduct testing [3]. Metamorphic testing involves multiple executions of the program under test. Outputs of these multiple executions and their computed outputs are expected to satisfy some necessary properties of the relevant algorithm if the implementation is correct. Such necessary properties are called metamorphic relations [4].

A. Metamorphic Relation

Metamorphic testing generates follow-up test cases by making reference to "metamorphic relations"(MR). For program p, an MR is a property of its target function f . The unique character of MR is that it involves multiple executions [5, 8]. A metamorphic relation, is an existing or expected relation over a set of distinct inputs and their corresponding outputs for multiple executions of the target function. For example, consider two inputs cos 16.3° and cos 376.3°. When we have to implement the program for cos, we have in knowledge that cos 16.3° = cos 376.3°. Because cos 376.3° = cos 360° + cos 16.3°. And cos function have periodic nature with respect to 360°.

B. Concept of oracle

An oracle is a mechanism against which people can decide whether the outcome of the program on test cases is correct. An oracle is a mechanism used by software testers and software engineers for determining whether a test has passed or failed. It is used by comparing the output(s) of the system under test, for a given test case input, to the outputs that the oracle determines that product should have.

Common oracles include:

- Specifications and documentation.
- An oracle for a software program might be a second program that uses a different algorithm to evaluate the same mathematical expression as the product under test.
- A consistency oracle that compares the results of one test execution to another for similarity
- A human being's judgment (i.e. does the program "seem" to the user to do the correct thing?

III. Why Trignometry?

There are an enormous number of uses of trigonometry and trigonometric functions. For instance, the technique of triangulation is used in astronomy to measure the distance to nearby stars, in geography to measure distances between landmarks and in satellite navigation systems [5,7]. Fields that use trigonometry or trigonometric functions include(on the oceans, in aircraft, and in space) acoustics, optics, electronics, probability theory, statistics, biology, medical imaging (CAT scans and ultrasound), pharmacy, chemistry, number theory (and hence cryptology), seismology, meteorology, oceanography, many physical sciences, land surveying and geodesy, architecture, phonetics, economics, electrical engineering, mechanical engineering, civil engineering, computer graphics, cartography, crystallography and game development [5,6]. And the most common way to test trigonometric component is to put values and verify the output with the expected value. This still doesn't make sure that the function will deliver the accurate result. To overcome this we use metamorphic testing [7].

IV. Metamorphic Testing of Trignometric Function

We will first insert a mutant in trigonometric function and then test the trigonometric function using special value testing and also by using metamorphic testing, and will verify the results that which technique is able to find the faulty program. We use a well known function of trigonometry i.e a cosine function. Consider a program p which implements the cosine function, cos(θ). The cos function f has a number of well known special test formulas that can be used in the testing of p. Test data and corresponding expected results are expressed in form of (θ, cos(θ)). Special values used in the testing are the following elements {(0, 0), (π/6, √3/2), (π/4, √2/ 2), (π/3, 1 /2), (π/2, 0)}.

We will take a program P which exactly calculates cosine function. And will insert a fault in program, and the faulty program will be called as FP. The faulty program FP will be used to examine that special value testing is not adequate for testing the cosine function. The fault introduced in the program is a little variation in the program. In order to verify the program correctness metamorphic testing is implemented using metamorphic relations. Various metamorphic relations used in this program are:

A) cos(x) = cosf(x)
B) cos(x) = cos(x+2*π)
C) cos(x) = -cos(X + π)
D) cos(x) = cos(2*π–x)
E) cos(x) + cos(y) + cos(z) – cos(x+y+z) =
2*Cos ((x+y)/2) * cos ((x-z)/2) + cos (π/2- (x+y+2z)/2) * cos (π/2- (x+y)/2)
F) cos(x) * cos(x) = 1-cos(π/2-x) * cos(π/2-x)
G) cos(-x) = cos(x)

Although there are numerous faults that could be introduced into the program, some of the faults produce errors that can easily be detected using special test values in testing and hence has little value in terms of demonstration. The following faulty program, however, produces errors that cannot be detected through the use of the defined special values and hence is considered a suitable candidate for demonstration purpose. Table 1 shows the result of faulty program executed after inserting the fault in the program and tested using special value testing and metamorphic testing. The first column lists the input test data set that includes special test values as well as random test values. The second column, fp(x) = expected result, is the result of testing fp using special values to verify against expected results. All other columns from third onwars (from A to G) are the results of testing by using metamorphic relations. Metamorphic testing does not require verification of the output of the sine function, that is cos(θ). The result is a verification of the inherent relationship to yield true or false as a result of examining the relationship A-G by multiple executions of program p. Since a metamorphic relation should always hold true for the given input domain D where test data set T⊂D, therefore the program is faulty if any of the metamorphic test fails. An entry of T in the table indicates that the relation holds true and F indicates that the relation does not hold true Computations of fp(x) using random test values as inputs cannot be verified using special value testing. The results are listed to show that had these random values been used, the fault in fp ( ) could not have been revealed. From the test results (shown in Table 1), a number of very useful observations can be made. When special test values are used to test the faulty program fp, all tests yield the expected results. In other words, the fault that has deliberately been seeded in the faulty program fp cannot be detected using the list of special values. But through metamorphic testing, errors can be uncovered using special values as test input. Metamorphic testing can be carried out using random test values in addition to special test values. Testers may freely select inputs randomly to test relation A-G without having to have expected results for cos(θ). In a sense, metamorphic testing provides a self-test mechanism based on the inherent metamorphic relationship [1].

A single metamorphic relation may not sufficiently test fp, for instance, A alone cannot detect any error using any of the special values and random values. E detects errors with all test cases. It uses the cosine function seven times in the relation and the repeated invocations of the same function with different parameters provide a higher chance of uncovering program faults since each invocation with a different parameter could have different execution paths. Relation A has not detected any error, whereas B detected errors two number of times, in C case the error has been detected 4 times, whereas in case E all the test cases detect errors. The implication is that a metamorphic relation with more invocations of the function is more likely to detect errors.

On the other hand, given a specific fault, the ability of each metamorphic relation to detect the error is still depended on the nature of the fault and the test cases. Metamorphic testing uses a black box testing approach and it is not known before testing is performed as to what type of errors it might detect, therefore it is useful to perform metamorphic testing with all available metamorphic relations. Finally, metamorphic testing has the flexibility to use random test values as input and not be restricted to only using special values.

V. Conclusion

This paper demonstrates the use of metamorphic testing. Metamorphic testing can reveal faults where special value testing cannot. The unique character of metamorphic testing is that it does not require human involvement to generate follow-up test cases and verify the test results and hence, it can be fully automated. We have also highlighted several important issues that are critical to the fault detection effectiveness of metamorphic California (2004). testing. Like other testing approaches, metamorphic testing only demonstrates the presence but not the absence of faults. In other words, metamorphic testing does not prove the correctness of the program and so it should be used in addition to other testing methods such as special value testing. In conclusion, the objective of this paper is to demonstrate that when special test values cannot sufficiently test a program, metamorphic testing could provide an effective way to complement the testing where test oracle is lacking. This has been achieved by using simple programs with strong metamorphic relationships.

Table 1 :

FP(x) ecpected result = Special test value		A	B	C	D	E	F	G
0	T	T	T	F	T	F	T	T
π/6	T	T	F	F	F	F	F	T
π/4	T	T	T	F	F	F	F	F
π/3	T	T	F	F	F	F	F	F
π/2	T	T	T	T	T	F	T	T

References

[1] Tsong Yueh Chen, Fei Ching Kuo, Ying Liu and Antony Tang, "Metamorphic Testing and Testing with Special Values",
[2] T.Y. Chen, F.C. Kuo, T.H. Tse, Zhi Quan Zhou, "Metamorphic Testing and Beyond", Computer Society Press, Los Alamitos,
[3] Zhi Quan Zhou k, D.H. Huang, T.H. Tse, Zongyuan Yang, Haitao Huang, T.Y. Chen, "Metamorphic Testing and Its Applications", 8th International Symposium on Future Software Technology (ISFST 2004).
[4] W. K. Chan, T. Y. Chen, Heng Lu, "Integration Testing of Context-Sensitive Middleware-Based Applications: a Metamorphic Approach", International Journal of Software Engineering and Knowledge Engineering.
[5] K.Y. Sim, W.K.S. Pao, C. Lin, "Metamorphic Testing Using Geometric Interrogation Technique and Its Application"
[6] Arnaud Gotlieb, Bernard Botella, " Automated Metamorphic Testing".
[7] W. K. Chan, W. K. Chan, Karl R. P. H. Leung, " A Metamorphic Testing Approach for Online Testing of Service-Oriented Software Applications.
[8] Sami Beydeda, "Self-Metamorphic-Testing Components.
[9] Tsong Yueh Chen, "Metamorphic Testing: A Simple Approach to Alleviate the Oracle Problem", 2010 Fifth IEEE International Symposium on Service Oriented System Engineering.