Scientific Software Testing : Creating and Invoking Metamorphic Functions

Sagar Lakhanotra
Computer Science Engineering
Institute of Technology, Nirma University
Ahmedabad, India
20bce511@nirmauni.ac.in

Het Mamtora
Computer Science Engineering
Institute of Technology, Nirma University
Ahmedabad, India
20bce513@nirmauni.ac.in

Unnat Mistry
Computer Science Engineering
Institute of Technology, Nirma University
Ahmedabad, India
20bce515@nirmauni.ac.in

***Abstract* — When input is altered in a predetermined manner, the result changes, and this is known as a metamorphic function. In this research, we provide a method for metamorphic function invocation and creation during scientific software testing. Metamorphic testing is, in general, a technique for automating the testing of diverse applications.**

Keywords — Metamorphic Testing, Scientific Testing, Software Testing, Scientific Software Testing, Metamorphic Functions, Invoking Metamorphic Functions.

I. INTRODUCTION – METAORPHIC TESTING

Software testing is a necessary but expensive activity used during software development to find programming errors. Executing a programme with test inputs during testing entails looking for errors, so there must be a process by which testers can determine whether the output of the programme is accurate or not. A method called metamorphic testing was developed to solve the oracle problem. Its foundation is the notion that it is frequently easier to reason about relationships between a program's outputs than it is to completely comprehend or formalise its input-output behaviour. The standard illustration is a programme that calculates the sine function: What does sin(12) actually equal? Is the output of 0.5365 as seen accurate? We can use this to evaluate whether $\sin(12) = \sin(\pi - 12)$ without knowing the specific numbers of each sine calculation since $\sin(x) = \sin(\pi - x)$ is a mathematical property of the sine function. This is an illustration of a metamorphic relation: an output relation that contrasts the results of two test cases, and an input transformation that can be used to create new test cases from test data that already exists. Not only can metamorphic testing solve the oracle problem, but it may also be very automated.

II. METAMORPHIC TESTING

Identity relations are a well-known concept in testing, and have been used even before the introduction of metamorphic relations. For example, Blum et al. checked whether numerical programs satisfy identity relations such as $F(x) = F(x1) + F(x2)$ for random values of x1 and x2. In the context of fault tolerance, the technique of data diversity runs the program on re-expressed forms of the original input; e.g., $\sin(x) = \sin(a) \times \sin(\pi/2 - b) + \sin(\pi/2 - a) \times \sin(b)$ where $a+b = x$. These notions from identity relations are generalised to any sort of connection, including equalities, inequalities, periodicity features, convergence restrictions, subsumption relationships, and many more, under Chen's concept of metamorphic testing, which he developed in 1998. A metamorphic connection is often stated as a relationship between a number of function inputs, $x1, x2, \ldots, xn$ (with $n > 1$), and their corresponding output values $f(x1), f(x2), \ldots, f(xn)$. For instance, the relationship between, as in the sine example from the introduction x1 and x2 would be $\pi - x1 = x2$, and the relation between f(x1) and f(x2) would be equality, i.e.:

$R = \{(x1, x2, \sin x1, \sin x2) \mid \pi - x1 = x2 \rightarrow \sin x1 = \sin x2\}$

The fundamental steps involved in using metamorphic testing may be summed up as follows:

A. *Costruction of Metamorphic Relations*

Determine the program's essential features, and then describe them as metamorphic relationships between various test case inputs and their anticipated outcomes. You should also consider how to create a follow-up test case from a source test case. Be aware that any preconditions that limit the source test cases to which metamorphic relations can be applied may be connected with them.

B. *Generation of Source Test Cases*

Create or choose a collection of source test cases using any conventional testing method for the software being tested (e.g., random testing).

C. *Execution of Metamorphic Test Cases*

Create follow-up test cases using the metamorphic relations, run the source and follow-up test cases, and inspect the relations. The metamorphic test case is said to have failed, indicating that the software under test has a defect, if the results of a source test case and its follow-up test case do not conform to the metamorphic relation.

III. STATE OF ART IN METAMORPHIC TESTING

The important contributions to metamorphic testing in the literature review are summarised below. The creation of metamorphic relations, the creation of source test cases, and the execution of metamorphic test cases are some of the ways that are categorised in accordance with the phase that was provided to the literature study for metamorphic testing process.

Properties of good metamorphic relations :

Designing efficient metamorphic relations is therefore a crucial step when applying metamorphic testing because the effectiveness of metamorphic testing is highly dependent on the specific metamorphic relations that are used. Numerous metamorphic relations with various fault-detection capabilities can be found for the majority of issues. To effectively test a given programme, it is therefore advised to use a variety of diverse metamorphic relations. Even more, some authors advise testing with as many metamorphic relations as you can. However, because defining metamorphic relations can be challenging, it's critical to understand how to pick the best ones. In this section, we examine papers that examine the characteristics of metamorphic relations that make them effective fault-finding tools.

1) Understanding the issue area is necessary to define good metamorphic relations.

2) Execution of the follow-up test case should alter as little as feasible from the source test case due to metamorphic relations.

3) Metamorphic relations originating from particular system components are more potent than those aiming at the entire system.

4) Formal descriptions of metamorphic relations are required.

Following flow chart elaborates how the metamorphic testing actually works with a proper flow.

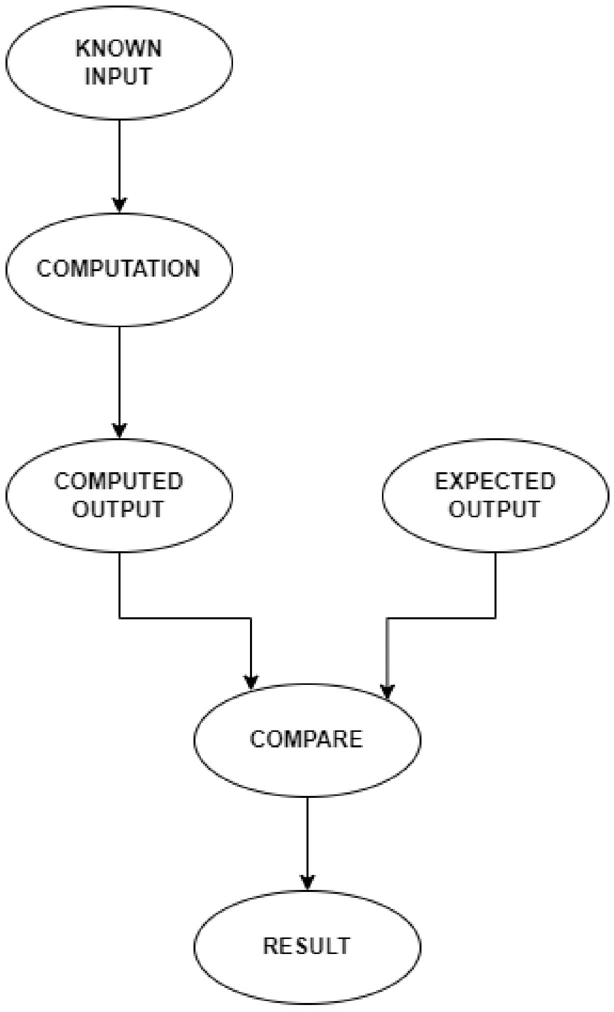

Fig 1. Metamorphic Testing Flow

IV. APPLICATION OF METAMORPHIC TESTING

We shall categorise different metamorphic testing application domains in this section. We found over 12 distinct application areas in all. Web services and applications (16%), computer graphics (12%), simulation and modelling (12%), and embedded systems (10%) are the most popular domains. Additionally, we discovered a wide range of uses in other industries (21%), including financial software, optimization tools, and encryption software.

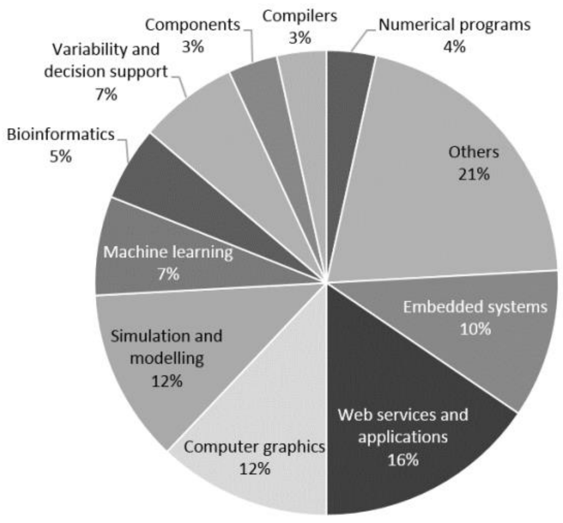

Fig 2. Metamorphic Testing Application Domains

The application domains for metamorphic testing are displayed in chronological sequence in Fig. 1. The (T)-designated domains were only theoretically investigated. As shown, the first use of metamorphic testing in the field of numerical programming was documented in 2002. The possible uses of metamorphic testing were primarily theorised about in the years that followed, but starting in 2007, there are applications in a variety of fields.

V. METAMORPHIC SOURCE TEST CASES

The utilisation of source test cases, which act as the starting point for the creation of subsequent test cases, is necessary for metamorphic testing.

Any conventional testing methods can be used to create source test cases. The findings are shown in Fig. 2 after we examined the various methods used in the literature and tallied the number of publications utilising each one. As shown, the bulk of studies (57%), followed by those that leveraged an existing test suite (34%), generated source test cases utilising random testing. Additionally, a number of articles (6%), including those that use constraint programming, search-based testing, or symbolic execution, also employ tool-based methodologies. The usefulness of metamorphic testing is supported by the variety of available sources. It also encourages the use of random testing as a simple, affordable method for creating the initial test suite.

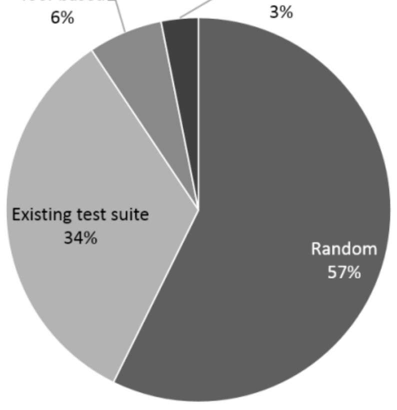

Fig 3. Source Test Case Generation Technique

VI. TYPES OF FAULTS

The success of metamorphic testing techniques is evaluated in terms of their capacity to identify errors in the programmes that are being tested that lead to failures. The main objective is to find actual bugs, however they aren't always accessible for analysis. As a result, most writers use tools for mutation testing to manually or automatically introduce false flaws (also known as mutants) into the subject programmes. Approximately 295 separate genuine faults have been found using metamorphic testing to date in 36 different tools, 23 of which are real world systems, indicating that metamorphic testing is effective at finding real flaws.

Metamorphic Relations

The test coverage of your ML application may be greatly increased by using metamorphic relations. "Refers to the relationship between the software input change and output change," says a metamorphic relation.

Evaluation Metrics

There have been several suggested measures to gauge how well metamorphic testing methods work. We chose two of them that were used so frequently in the publications under review that they may be regarded as the de facto norms for metamorphic testing research. Below are the two of the measures explained in detail to understand the evaluation metrics of the mutation metrics in the proposed scientific software testing with various invoked metamorphic functions.

A. Mutation Score

This statistic is based on mutation analysis, in which mutation operators are used to create copies of the programme being tested that contain fake flaws (mutants). The proportion of discovered ("killed") mutants to all mutants is known as the mutation score. Equivalent mutations are those that do not alter the semantics of the programme and are hence undetectable. Despite the fact that programme equivalence is debatable, in reality it is not always possible to eliminate comparable mutants from the overall number of mutants. Consider a collection of metamorphic tests, or pairs of source and follow-up test cases, that make up a metamorphic test suite. The following formula is used to determine t's Mutation Score (MS):

$$MS(t) = \frac{M_k}{M_t - M_e}$$

where Mt is the total number of mutants, Me is the number of equivalent mutants, and Mk is the number of mutants killed by the metamorphic tests in time (t). The ratio of mutants identified by a certain metamorphic relation r is frequently calculated using a variation of this measure as follows:

$$MS(t,r) = \frac{M_{kr}}{M_t - M_e}$$

where Mkr is the number of mutants destroyed during the t-derived from-r metamorphic testing. The mutation detection ratio is another name for this measure.

B. Fault Detection Ratio

This measure determines the proportion of test cases that find a certain flaw. The formula for calculating the Fault Detection Ratio (FDR) of a metamorphic test suite and a fault is as follows:

$$FDR(t,f) = \frac{T_f}{T_t}$$

where Tt is the total number of tests in t, and Tf is the number of tests that detect f. Using a specified metamorphic relation r, one variation of this measure determines the proportion of test cases that identify a flaw as follows:

$$FDR(t,f,r) = \frac{T_{fr}}{T_r}$$

Where Tr is the total number of metamorphic tests generated from r, and Mf r is the number of tests in t derived from the relation r that identify the flaw f. Fault finding rate is another name for this measure.

VII. IMPLEMENTATION CODE LOGIC

Following is a logic of code segment presented for the better understanding of created and invoked metamorphic functions. This logic concludes the test results for the desired input.

Metamorphic relations for binary search for Array A, index k and respective element x at index k:

1. if x = A[k], then binary_search(x, A) = k
2. if A[k-1] < x < A[k+1] and x != A[k], then binary_search(x, A) = -1
3. if x = A[k], then binary_search(A[k-1], A) = k-1 and binary_search(A[k+1], A) = k+1

Metamorphic relations for kth_occurence for Array A, element x and respective element of index k;

1. if kth_occurrence(x, k, A) = i, then A[i] = x
2. kth_occurrence(A[i], 1, A) != -1

CONCLUSION

The above research work findings demonstrate the popularity of metamorphic testing and the rising trend of contributions to the field. Additionally, we discovered evidence of the technique's integration with other testing methodologies and its use in a variety of areas that go well beyond numerical programming. All of the research points to metamorphic testing as a maturing testing approach that may be used to generate test data automatically as well as to solve the oracle problem. Finally, despite the improvements in metamorphic testing, our review identifies several unexplored regions.

REFERENCES

[1] A Survey on Metamorphic Testing - Sergio Segura, Gordon Fraser, Ana B. Sanchez, and Antonio Ruiz-Cortes

[2] H. Liu, F.-C. Kuo, D. Towey, and T. Y. Chen, "How effectively does metamorphic testing alleviate the oracle problem?" Software Engineering.

[3] Z. Hui and S. Huang, "Achievements and challenges of metamorphic testing,"

[4] X. Xie, J. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, "Application of metamorphic testing to supervised classifiers,"

[5] Y. Yao, C. Zheng, S. Huang, and Z. Ren, "Research on metamorphic testing: A case study in integer bugs detection,"

[6] X. Xie, J. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, "Application of metamorphic testing to supervised classifiers,"

[7] T. Jameel, L. Mengxiang, and C. Liu, "Test oracles based on metamorphic relations for image processing applications,"

[8] C. Castro-Cabrera and I. Medina-Bulo, "An approach to metamorphic testing for ws-bpel compositions,"

[9] G. Dong, T. Guo, and P. Zhang, "Security assurance with program path analysis and metamorphic testing,"

[10] U. Kanewala and J. M. Bieman, "Using machine learning techniques to detect metamorphic relations for programs without test oracles,"