Metamorphic Fault Tolerance: An Automated and Systematic Methodology for Fault Tolerance in the Absence of Test Oracle

Huai Liu, Iman I. Yusuf, Heinz W. Schmidt
Australia-India Research Centre for Automation Software Engineering
RMIT University, Melbourne, Australia
{huai.liu, iman.yusuf, heinz.schmidt}@rmit.edu.au

Tsong Yueh Chen
Department of Computer Science and Software Engineering
Swinburne University of Technology, Melbourne, Australia
tychen@swin.edu.au

August 6, 2014

Abstract

A system may fail due to an internal bug or a fault in its execution environment. Incorporating fault tolerance strategies enables such system to complete its function despite the failure of some of its parts. Prior to the execution of some fault tolerance strategies, failure detection is needed. Detecting incorrect output, for instance, assumes the existence of an oracle to check the correctness of program outputs given an input. However, in many practical situations, oracle does not exist or is extremely difficult to apply. Such an oracle problem is a major challenge in the context of software testing. In this paper, we propose to apply metamorphic testing, a software testing method that alleviates the oracle problem, into fault tolerance. The proposed technique supports failure detection without the need of oracles.

keywords: Fault tolerance, oracle problem, metamorphic testing, metamorphic relation.

1 Introduction

Software reliability is the probability of a system performing its normal function for a period of time in a given environment [3]. One major threat to reliability is *failure*, which is the behavior of a system that deviates from normal system functions. The root cause of a failure is a fault, a defect in the system. Fault

removal and fault tolerance, for instance, are used to improve the reliability of a system [3]. Fault removal aims to remove as many existing faults of a system as possible. Software testing techniques can be applied to reveal software faults. However, it is extremely difficult, if not impossible, to remove all program faults. Fault tolerance, on the other hand, is concerned with enabling a system to perform its normal function despite the presence of faults.

The execution of a reactive fault tolerance strategy is preceded by failure detection. Detecting a failure like incorrect output, which is a main interest of this paper, requires a mechanism for checking the correctness of a program output. Such an output verification mechanism is termed as *oracle* in the context of software testing. Only with an oracle can we decide whether a failure occurs, and thus whether we should apply the fault tolerance strategy to provide an alternative program output. When the oracle does not exist or is extremely difficult to apply (namely the *oracle problem* in the context of testing), the applicability and effectiveness of a fault tolerance strategy will be significantly affected. One way of addressing the oracle problem within the context of fault tolerance is by using assertions to detect the failures due to the violation of certain properties (such as, out of range, incorrect data type, etc) [14].

The oracle problem is not rare in practice, and has become a major research focus in software testing. Metamorphic testing [8] aims at addressing the oracle problem. It makes use of some relations among multiple inputs and their corresponding outputs, namely *metamorphic relations*, to provide a test result verification mechanism alternative to the oracle. Previous studies [6] have shown that metamorphic testing can detect more faults than assertion checking [10], another popular technique for alleviating the oracle problem. A more recent study [8] demonstrated that metamorphic testing can effectively alleviate the oracle problem, and a small number of diverse metamorphic relations are sufficient by themselves to imitate a test oracle in terms of fault detection.

In this paper, we discuss how to apply the basic intuition of metamorphic testing into fault tolerance, and thus propose the theoretical framework of a new technique that can handle system failure without the need of oracles during failure detection. Various research questions will be discussed, and proposals for the research directions will be made.

2 Metamorphic testing

Metamorphic testing is executed in the following steps. First, testers analyze the software under test and identify some necessary properties, which are then represented in the form of relations among multiple inputs and their corresponding outputs, namely metamorphic relations. Secondly, some test cases, referred to as the source test cases, are generated using some traditional test case generation techniques (such as random testing [9]). Thirdly, new test cases, termed as the follow-up test cases, are constructed from the source test cases and based on the metamorphic relations. After both source and follow-up test cases are executed on the software under test, their outputs are verified against the meta-

morphic relations. If a metamorphic relation is violated, a failure is said to be detected.

The basic process of metamorphic testing is illustrated by the following example. Suppose that a web search engine S takes a set of key words as an input and displays the results containing all the key words. The following two metamorphic relations can be identified for S.

MR_a if the order of the key words is changed, the search results should remain unchanged.

MR_b if one key word is deleted, the amount of search results should either increase or at least remain unchanged.

Suppose that we have generated a source test case t containing a set of key words, and its associated output is o containing a set of search results. According to MR_a, we can generate a follow-up test case t' by simply permutating t. Given that the output associated with t' is o'. We need to check whether the relation $o' = o$ is satisfied or violated. If violated, a failure is said to be detected. Similarly, according to MR_b, we can have another follow-up test case t'' by removing one key word from t. After the execution of test cases, we check whether o'' (the output of t'') and o hold the relation $o \subseteq o''$.

3 Theoretical framework

To address the oracle problem in software fault tolerance, we propose a new technique, namely *metamorphic fault tolerance*. In the context of testing, metamorphic relations are used for (i) test case generation (especially for follow-up test cases) and (ii) test result verification. Our new technique makes use of metamorphic relations in the following three aspects.

Generator Based on the original input t, generate other inputs t', t'', $\cdots$ according to metamorphic relations.

Checker Through the checking against the metamorphic relations among the outputs of t, t', t'', $\cdots$ (denoted by o, o', o'', $\cdots$, respectively), judge whether or not o is trustworthy.

Calculator If o is untrustworthy, calculate the output from o', o'', $\cdots$ based on the information provided by the metamorphic relations.

Figure 1 shows the basic process of metamorphic fault tolerance. In the framework, the generator implements a similar function to the follow-up test case generation in metamorphic testing: It makes use of metamorphic relations to generate other new inputs. In particular, it should be pointed out that the generator is not limited to the conversion of t into other inputs — the inputs that are converted from t can be further transformed into other inputs. For example, t can be converted into t' according to a metamorphic relation. The same or a different metamorphic relation can be applied to further convert t'

into another input; in other words, inputs can be iteratively generated based on a set of metamorphic relations [12]. Note that in metamorphic testing, multiple executions are required to verify the test results. Similarly, metamorphic fault tolerance also executes the system using the generated multiple inputs in parallel for output verification. Though the parallel executions of the system may be expensive, the cost would be acceptable for systems that require high reliability. Generating new inputs (or follow-up test cases) has been extensively discussed in previous studies of metamorphic testing [8, 12]. Thus we mainly focus on checker and calculator) in this paper.

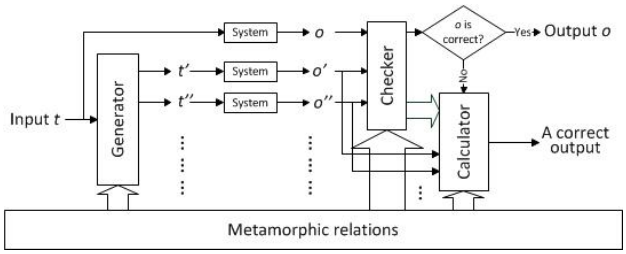

Figure 1: Framework of metamorphic fault tolerance

3.1 Checker

The checker in metamorphic fault tolerance is more complicated than the test result verification process in metamorphic testing. A major objective of testing is to detect failures. In metamorphic testing, once a metamorphic relation is violated, a failure is said to be detected, no matter whether the failure is actually caused by the source or the follow-up test case, or even both. However, in metamorphic fault tolerance, when a metamorphic relation is violated, it is also required to know whether or not the original input t is the reason of the violation, based on which we can in turn decide whether or not its output o can be used.

A checking algorithm should be carefully designed to judge whether t causes a failure or results in a trustworthy output. Intuitively speaking, our confidence on the trustworthiness of t depends on the number of violations or satisfactions of metamorphic relations. The more relations are satisfied and the less relations are violated, the more trustable t is. Let us recall the search engine S mentioned in Section 2. Suppose that MR_a and MR_b are the only two metamorphic relations identified for S. If o and o' violates MR_a, o and o'' violates MR_b, but o' and o'' satisfies MR_b (note that t' and t'' can also be used as the source and follow-up test cases in MR_b), it is very likely that the input t is untrustworthy. It should be pointed out that research has been conducted on how to check the trustworthiness of different inputs based on metamorphic relations [5]. We can make use of the results of these previous studies in our work of fault tolerance.

3.2 Calculator

The calculator is a new mechanism that has never been investigated in previous studies in metamorphic testing, which is mainly focused on showing the presence of faults but not on providing the acceptable outputs. In fault tolerance, after the failure detection, some mechanisms should be applied for fault recovery, for example, alternative outputs should be computed. Luckily, the checking against the metamorphic relations may provide some useful information for us to get the trustworthy output as the alternative to the failed original output. If t is judged as untrustworthy by the checker, we can further evaluate the trustworthiness of each new input (t', t'', $\cdots$), decide the most reliable input, say t_b, and then produce the most trustworthy output based on the output (o_b) associated with t_b. However, metamorphic relations are not limited to equivalence ones — actually, any form of relation is acceptable in metamorphic testing as long as the relation can be checked (for instance, MR_b in Section 2 involves the relation of "subset"). Therefore, the production of the trustworthy output is not so straightforward as it looks. Let us consider the following scenarios.

Case 1 There exists a metamorphic relation that two inputs are associated with the identical outputs. MR_a in Section 2 is a typical example for such a relation. For this case, given the most reliable input t_b, its associated output o_b can be directly used for the system.

Case 2 Instead of the exact equivalence relation, there exists a metamorphic relation that involves an equation, which can in turn help us calculate a trustworthy output based on o_b. For example, suppose that a metamorphic relation is that for two inputs t and t_b, $o_b = 2 \times o + 1$. If o is judged as untrustworthy, we can calculate the output as $(o_b - 1)/2$.

Case 3 If it is impossible to get the exact answer based on only one metamorphic relation, another metamorphic relation(s) may be used to pose some limitations such that we can make the best choice. Suppose a metamorphic relation is that for two inputs t and t_b, $o_b = o^2$. After a simple calculation, we have two options, $\sqrt{o_b}$ and $-\sqrt{o_b}$, for the system output. If there is another metamorphic relation that for two inputs t and t', $o' > o$ and $o' < \sqrt{o_b}$, we can then decide that the output should be $-\sqrt{o_b}$.

Case 4 If all the above three cases are impossible, we can still choose another input, say t_c, which is also more trustworthy than t though it may not be as trustworthy as t_b. If t_c falls into one of the Cases 1 to 3, we can tolerate the system fault based on t_c's output.

Note that we use simple scientific functions to illustrate the above various scenarios for metamorphic fault tolerance. In practice, metamorphic relations can be of more complex forms, decided by the innate software properties, and thus be applied in the testing of various application domains (not limited to mathematical properties), such as office applications [6], financial services [11], telecommunications [4], etc. Thus, the implementation of calculator will depend

on the nature of the software system. Moreover, the calculator of a trustworthy output may become expensive if a large number of inputs and metamorphic relations are involved. Greedy algorithms may be applied to reduce the computation overhead. It should also be pointed out that there exist situations where no trustworthy alternative output can be provided, that is, none of the above four cases is valid. Under such situations, approximate output might be provided or even an error report might be issued.

In summary, the rich information provided by metamorphic relations not only helps to check the correctness of the system output more precisely, but also provides a more effective mechanism for obtaining a trustworthy output.

4 Discussion

Besides various operations of the checker and the calculator, the following two are also very important to our research:

4.1 Identification of Metamorphic Relations

As shown in Figure 1, metamorphic relations are the core part of the new technique. The mechanism to identify high-quality metamorphic relations has been one important research topic in metamorphic testing, and it was suggested in the most recent study [8] that the diversity among metamorphic relations is the key factor for ensuring a high testing effectiveness. Diversity should also be highly recommended for identifying metamorphic relations in this new fault tolerance technique. It has been widely acknowledged that similar program inputs tend to cause similar execution behaviors and thus show a high degree of similarity in failure detection. Given such intuition, metamorphic relations involving similar inputs may not be effective in precisely finding the failure-causing input. In a word, a high degree of diversity among metamorphic relations is very important for providing a trustworthy judgment among different inputs as well as a reliable calculation of outputs.

Besides diversified, the identified metamorphic relations should also be evenly related to each possible input. Suppose that there is a bias in the metamorphic relations; for instance, most of them are correlated with certain part of inputs. As a result, the checking process will be unfair for those inputs that are not extensively covered by the metamorphic relations: Even if these inputs provide acceptable outputs, they will still get fewer satisfactions because they are only associated with a small number of metamorphic relations. Briefly speaking, when identifying metamorphic relations for the new technique, the number of relations correlated to different types of inputs must be evenly distributed across the whole input space of the software system.

In addition, the identified metamorphic relations should be as tight as possible in order to increase the probability of obtaining an exact result. For example, MR_b can be enhanced to the following metamorphic relation:

MR_c Given two inputs t and t', where t' is constructed by removing one key word from t, their associated outputs o and o' should have the following relation: $o' \backslash o = l_k$, where l_k is the set of results that are in o' but do not contain the removed key word.

MR_c is a tighter relation than MR_b, and thus can provide a more effective clue to calculate an exact result.

4.2 Comparison and Integration with Other Fault Tolerance Strategies

There exist several fault tolerance strategies that address the oracle problem to some extent. For example, the N-version programming technique [2] makes use of multiple implementations for the same system to handle failure. For the same input, all versions will be executed in parallel to provide a set of outputs, based on which a ranking algorithm is then implemented to decide the majority of outputs, if there is no test oracle. However, there are some criticism on the effectiveness of such a method in alleviating the oracle problem. Knight and Leveson [7] conducted a series of experiments and found that even if different teams were recruited to develop multiple versions, they might make similar mistakes and consequently caused similar faults. In such a situation, the ranking algorithm in the N-version programming may not give the correct answer.

Another fault tolerance strategy similar to our technique is data diversity [1], which makes use of multiple inputs that are expected to have the same output. Once an input causes a failure, it is "reexpressed" into a new input, which hopefully can provide the acceptable output.Though data diversity is very similar to the simplest case (Case 1 in Section 3.2) in metamorphic fault tolerance, we would like to point out that data diversity is fundamentally different from metamorphic fault tolerance. Data diversity has a constraint: only the equivalence relation is allowed. On the contrary, metamorphic relations can be of any form. As a result, data diversity can only work when Case 1 in Section 3.2is valid; while metamorphic fault tolerance is applicable under various scenarios due to the rich forms of metamorphic relations.

A promising research direction is to integrate the new metamorphic fault tolerance technique with these existing strategies. For instance, using metamorphic relations in the N-version programming can help more precisely checking which version provides a reliable output given certain inputs, and thus increasing the correctness of its ranking algorithm. Moreover, metamorphic relations can enrich the data diversity across the input space, and thus can deliver a more effective fault tolerance support. The research on such integration will result in a family of fault tolerance strategies without the need of oracles.

5 Conclusion

Software faults are almost unavoidable despite extensive testing and debugging. In order to enable software systems to perform their functions, various fault tol-

erance strategies have been proposed. The execution of these strategies involves failure detection. However, detecting failures, in particular incorrect outputs, implicitly assumes the existence of an oracle, which is a mechanism to check whether an input results in a correct output or causes a software failure. In this paper, we proposed a new technique, namely metamorphic fault tolerance, which handles system failure without the need of oracles during failure detection. Metamorphic relations, which are originally used in alleviating the oracle problem in the context of software testing, are leveraged in fault tolerance to help checking whether or not an input causes a failure as well as calculating the acceptable output. A theoretical framework of the new technique has been presented. Also discussed were the strategies on how to properly conduct the research on the new technique.

Metamorphic testing has been used to alleviate the oracle problem in several areas, such as program proving and debugging, and thus has resulted in novel techniques in the areas, namely semi-proving [5] and metamorphic slicing [13], respectively. These techniques basically are the application of metamorphic testing into areas related to fault removal. The new technique is the first systematic approach to addressing the oracle problem in the field of fault tolerance. It would be interesting to investigate whether and how metamorphic testing can be applied into other reliability engineering fields, such as fault prevention and fault forecasting. As the first paper on metamorphic fault tolerance, we used simple examples to illustrate the feasibility and applicability of our framework. In fact, different types of metamorphic relations have been identified and used for the testing of various systems, such as wireless network [4], web services [11], etc. There is no reason why these types of metamorphic relations could not be used for the fault tolerance in the specific fields; however, this needs to be assessed through further studies.

6 Acknowledgments

This research was supported by an Australia Research Council Discovery Grant (DP120104773).

References

[1] P. E. Ammann and J. C. Knight. Data diversity: An approach to software fault tolerance. *IEEE T Comput*, 37(4):418–425, 1988.

[2] A. Avizienis. The N-version approach to fault-tolerant software. *IEEE T Software Eng*, 11(12):1491–1501, 1985.

[3] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE T Depend Secure*, 1(1):11–33, 2004.

[4] T. Y. Chen, F.-C. Kuo, H. Liu, and S. Wang. Conformance testing of network simulators based on metamorphic testing technique. In *Proceedings of FORTE'09*, pages 243–248, 2009.

[5] T. Y. Chen, T. H. Tse, and Z. Zhou. Semi-proving: An integrated method for program proving, testing, and debugging. *IEEE T Software Eng*, 37(1):109–125, 2011.

[6] P. Hu, Z. Zhang, W. K. Chan, and T. H. Tse. An empirical comparison between direct and indirect test result checking approaches. In *Proceedings of SOQUA'06*, pages 6–13, 2006.

[7] J. C. Knight and N. G. Leveson. An experimental evaluation of the assumption of independence in multiversion programming. *IEEE T Software Eng*, 12(1):96–109, 1986.

[8] H. Liu, F.-C. Kuo, D. Towey, and T. Y. Chen. How effectively does metamorphic testing alleviate the oracle problem? *IEEE T Software Eng*, in press.

[9] G. J. Myers. *The Art of Software Testing*. John Wiley and Sons, second edition, 2004.

[10] D. S. Rosenblum. A practical approach to programming with assertions. *IEEE T Software Eng*, 21(1):19–31, 1995.

[11] C.-A. Sun, G. Wang, B. Mu, H. Liu, Z. Wang, and T. Y. Chen. Metamorphic testing for web services: Framework and a case study. In *Proceedings of ICWS'11*, pages 283–290, 2011.

[12] P. Wu. Iterative metamorphic testing. In *Proceedings of COMPSAC'05*, volume 1, pages 19–24, 2005.

[13] X. Xie, W. E. Wong, T. Y. Chen, and B. Xu. Metamorphic slice: An application in spectrum-based fault localization. *Inform Software Tech*, 55(5):866–879, 2013.

[14] I. I. Yusuf, H. W. Schmidt, and I. D. Peake. Architecture-based fault tolerance support for grid applications. In *Proceedings of QoSA'11*, pages 177–182, 2011.