Towards Metamorphic Testing of Concurrent Programs

Peng Wu
State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China, wp@ios.ac.cn

Hepeng Dai
School of Computer and Communication Engineering
University of Science and Technology Beijing
Beijing, China
daihepeng@sina.cn

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

Chang-ai Sun, *Member, IEEE*
School of Computer and Communication Engineering
University of Science and Technology Beijing
Beijing, China
casun@ustb.edu.cn

***Abstract*—Metamorphic testing (MT) is a promising technique to alleviate the oracle problem, which fist defines metamorphic relations (MRs) used to generate new test cases (i.e. follow-up test cases) from the original test cases (i.e. source test cases). Both source and follow-up test cases are executed and their results are verified against the relevant MRs.**

***Index Terms*—metamorphic testing, control test process, partition**

I. Introduction

Testing of a concurrent program, e.g., a multi-threaded program, is still a challenging task due to its nature of nondeterminism. Moreover, a concurrent program typically consists of multiple threads that cooperate closely to fulfill a functional assignment with high efficiency, e.g., computing, sorting, or searching a large volume of static or dynamic data. A test oracle [1] usually provides an exact mechanism of deciding whether an output produced by a program is correct or not. This oracle problem [2], [3] also arises for concurrent programs as it can be very difficult or expensive, if not impossible, to determine their expected outputs *a priori*.

Metamorphic testing (MT) [4], [5] has been a successful technique to tackle oracle problems in various domains. It first defines metamorphic relations (MRs) based on the necessary properties about the inputs and outputs of the program under test. Then, MRs are exploited in two ways: a new test case (i.e., a follow-up test case) can be generated from an original test case (i.e., a source test case) based on an MR; the outputs resulted by executing the source and follow-up test cases can be verified against the corresponding MR. If the MR is violated, the program under test is shown to be incorrect, as witnessed by the pair of the source and follow-up test cases.

The National Natural Science Foundation of China (under Grant No. 61872039), the Beijing Natural Science Foundation (Grant No. 4162040), the Aeronautical Science Foundation of China (Grant No. 2016ZD74004), and the Fundamental Research Funds for the Central Universities (Grant No. FRF-GF-17-B29).

Let us use a simple example to illustrate how MT works. For instance, consider the function $max(x, y)$ that returns the maximum value between two integers x and y. It enjoys a simple and obvious property that the order of the two parameters x and y shall not affect the outputs. This can be identified as the metamorphic relation $MR_{max} : max(x, y) = max(y, x)$, where (x, y) and (y, x) constitutes the source and the follow-up test case, respectively. Then, given a program $P(x, y)$ that implements the function $max(x, y)$, and a source test case $(1, 2)$, if the outputs of P running with this source test case and the follow-up test case $(2, 1)$ do not equal, i.e, $P(1, 2) \neq P(2, 1)$, a fault is detected in P.

In this paper we aim at investigating the applicability of metamorphic testing to concurrent programs. This is motivated by the observation that the functional requirement or user expectation to a concurrent program can be identified as metamorphic relations for verifying the outputs of its non-deterministic executions under different inputs.

The rest of the paper is organized as follows.

II. Metamorphic Testing of Concurrent Programs

A. Concurrent Programs

Consider a function $\textsc{top}(V, n)$ that returns in the ascending order the minimal n values in a list V, while V may contain duplicate elements. For a list of large size, a sequential search program may not implement this function in a highly efficient way. Many concurrent search structures have been proposed to take advantages of multi-processor architectures, which are already very popular nowadays. Concurrent implementations of the *priority queue* data structure are typically adopted to implement multi-threaded programs for the function $\textsc{top}(V, n)$. In this work, we use the 5 concurrent priority queue classes presented in the textbook [6]: SimpleLinear, SimpleTree, SequentialHeap, FineGrainedHeap, and SkipQueue.

The SimpleLinear class is an array-based bounded implementation, while the SimpleTree class is a tree-based one.

The SequentialHeap class is a coarse-grained implementation based on a unbounded heap, while the FineGrainedHeap class is a fine-grained one. These four classes all explicitly use locks for synchronization, while the SkipQueue class does not. It is based on a skiplist that uses atomic primitives (i.e., compareAndSet) for synchronization.

B. Metamorphic Relations

Let VW be the concatenation of lists V and W, and V^k the concatenation of k V's. Let $head(V)$ denote the first element of list V, and $tail(V)$ the list resulted by removing the first element of list V. Therefore, $V = head(V) :: tail(V)$. The higher-order function (map f V) applies the given function f to each element of list V, returning a list of the results in the same order, i.e., (map f V) = $f(head(V))$::(map f $tail(V)$).

Assume $\text{TOP}(V, n) = [y_1, \ldots, y_n]$ with $y_1 < \cdots < y_n$, where $y_1, \ldots, y_n$ are the minimal n values in list V. Then, metamorphic relations of $\text{TOP}(V, n)$ can be defined as follows.

1) Permutation: A permutation of V is a rearrangement of the elements of list V. Such rearrangement shall not affect the result of TOP. This property is identified as the following metamorphic relations, where *MR*$_2$ constitutes a special case of *MR*$_1$.

MR-1 $\text{TOP}(V', n) = \text{TOP}(V, n)$ for any permuation V' of V.

MR-2 (Commutative Law) $\text{TOP}(VW, n) = \text{TOP}(WV, n)$.

2) Insertion: Adding duplicate elements into list V shall not affect the result of TOP. This general property is identified as the following metamorphic relations, which choose duplicate elements from different perspectives.

MR-3 $\text{TOP}(V^k, n) = \text{TOP}(V, n)$ for any $k > 1$.

MR-4 $\text{TOP}(VV', n) = \text{TOP}(V, n)$ for any $V' \subseteq V$.

MR-5 $\text{TOP}(V[y], n)$=$\text{TOP}(V, n)$ for any y in $\text{TOP}(V, n)$.

MR-6 $\text{TOP}(VV_1 \cdots V_k, n) = \text{TOP}(V, n)$ for any $k \geq 1$, where $V_i \subseteq \text{TOP}(V, n)$ for every $1 \leq i \leq k$.

MR-7 $\text{TOP}(V[y_1, \ldots, y_n], n) = \text{TOP}(V, n)$.

MR-8 $\text{TOP}(V[y_n, \ldots, y_1], n) = \text{TOP}(V, n)$.

MR-9 $\text{TOP}(VW, n) = \text{TOP}(V, n)$ for any permutation W of $\text{TOP}(V, n)$.

MR-10 $\text{TOP}([y_1, \ldots, y_n]V, n) = \text{TOP}(V, n)$

MR-11 $\text{TOP}([y_n, \ldots, y_1]V, n) = \text{TOP}(V, n)$

MR-12 $\text{TOP}(WV, n) = \text{TOP}(V, n)$ for any permutation W of $\text{TOP}(V, n)$.

The following metamorphic relations add to list V fresh elements that are no greater than the last element y_n of $\text{TOP}(V, n)$. These elements shall occur in the follow-up result of TOP.

MR-13 $\text{TOP}(V[y], n) = [y_1, \ldots, y_i, y, y_{i+1}, \ldots, y_{n-1}]$ if $y \notin \text{TOP}(V, n)$ and there exists $1 \leq i < n$ such that $y_i < y < y_{i+1}$.

MR-14 $\text{TOP}(V[y'_1, \ldots, y'_k], n) = [y'_1, \ldots, y'_k, y_1, \ldots, y_{n-k}]$, where $k \leq n$ and $y'_1 < \cdots < y'_k < y_1$.

On the contrary, the following metamorphic relation *MR*-15 adds to list V fresh elements that are greater than y_n. This insertion shall not affect the result of TOP.

MR-15 $\text{TOP}(V[y''_1, \ldots, y''_k], n) = \text{TOP}(V, n)$, where $y_n < y''_1 \leq \cdots \leq y''_k$.

3) Deletion: Deleting an element from list V will change the result of TOP if it also occurs in $\text{TOP}(V, n)$. The follow-up result of TOP only differs from the original one in those elements deleted.

MR-16 Let $X = [x_1, \ldots, x_k]$ for arbitrary $k \geq 1$ elements $x_1, \ldots, x_k$, $V' = V \backslash X$, $X' = X \cap \text{TOP}(V, n)$.
- Then, $\text{TOP}(V, n) \backslash \text{TOP}(V', n) \subseteq X'$
- Specially, if $X' = \emptyset$, $\text{TOP}(V', n) = \text{TOP}(V, n)$;

where $V \backslash X$ removes from V only one occurrence of each element in X (if any).

4) Transformation: Transforming all the elements of list V will make the result of TOP transformed in the same way.

MR-17 TOP((map f V),n)=(map f TOP(V,n)). For example, $f(x) = x + c$ for any constant c.

5) Splitting: Suppose $V = V_1V_2$. If y is one of the minimal n values in list V, then it is also one of the minimal n values in list V_1 or V_2. This property is identified as the following metamorphic relation *MR*-18.

MR-18 $\text{TOP}(V, n) \subseteq \text{TOP}(V_1, n) \cup \text{TOP}(V_2, n)$.

If y is one of the minimal n values in list V_1, and also one of the minimal m values in list V_2, then it is one of the minimal $n + m$ values in list V. This property is identified as the following metamorphic relations, which differ only in m.

MR-19 $\text{TOP}(V_1, n) \cap \text{TOP}(V_2, n) \subseteq \text{TOP}(V, 2n)$.

MR-20 $\text{TOP}(V_1, n) \cap \text{TOP}(V_2, m) \subseteq \text{TOP}(V, n + m)$ for $1 \leq m < n$.

MR-21 $\text{TOP}(V_1, n) \cap \text{TOP}(V_2, m) \subseteq \text{TOP}(V, n + m)$ for $m > n$.

The concatenation of the results of TOP on V_1 and V_2 preserves the result of TOP on V. This property is identified as the following metamorphic relation *MR*-22.

MR-22 $\text{TOP}(V, n)$=$\text{TOP}(\text{TOP}(V_1, n)\text{TOP}(V_2, n), n)$.

6) Sublisting: $\text{TOP}(V, n)$ is a prefix of $\text{TOP}(V, m)$ if $n < m$, while $\text{TOP}(V, n)$ is an extension of $\text{TOP}(V, m)$ if $n > m$. Specially, we consider the boundary values of m in the following metamorphic relations.

MR-23 $\text{TOP}(V, n)$ is a prefix of $\text{TOP}(V, n + 1)$.

MR-24 $\text{TOP}(V, n)$ is a prefix of $\text{TOP}(V, m)$ if $m > n + 1$.

MR-25 $\text{TOP}(V, n)$ is an extension of $\text{TOP}(V, n - 1)$.

MR-26 $\text{TOP}(V, n)$ is an extension of $\text{TOP}(V, m)$ if $1 \leq m < n - 1$.

III. Empirical Study

In this section we present the research questions concerned in this work, and the case study on the concurrent implementations of $\text{TOP}(V, n)$. Then, we discuss the experimental results with detailed analysis.

A. Research Questions

RQ-1 How effective MT is at detecting faults of concurrent programs? What is the actual overhead for MT detecting these faults, in terms of time consumption and the number of test cases executed?

Fault-detection effectiveness and efficiency are key for evaluating the performance of a testing technique. In our study, we adopt five state-of-the-art concurrent priority queue classes to implement the function TOP(V, n) within various concurrent scenarios. Then, we apply mutation analysis to evaluate the performance of MT in detecting the faults seeded, especially the concurrency faults related to locks and atomic primitives.

RQ-2 What is the fault-detecting capability of an individual metamorphic relation in different concurrent scenarios? How are metamorphic relations relevant to programs faults, especially concurrency faults?

RQ-3 How many threads are sufficient for MT detecting a concurrency fault?

RQ-4 How many metamorphic relations are sufficient for MT detecting the faults that remain in a concurrent program? (probably not in this paper)

B. Experimental Results

1) *Mutants:*

2) *Concurrent Scenarios:*

3) *Response to RQ-1:*

4) *Response to RQ-2:*

5) *Response to RQ-3:*

IV. Related Work

In this section, we describe related work of MT.

A. Metamorphic Testing

When testing a software system, the oracle problem appears in some situations where either an oracle does not exist for the tester to verify the correctness of the computed results; or an oracle does exist but cannot be used. The oracle problem often occurs in software testing, which renders many testing techniques inapplicable [2]. To alleviate the oracle problem, Chen et al. [4] proposed a technique named metamorphic testing (MT) that has been receiving increasing attention in the software testing community [2], [5], [7]. The main contributions to MT in the literature focused on the following aspects: i) MT theory; ii) combination with other techniques; iii) application of MT.

1 *Theoretical development of MT:* The MRs and the source test cases are the most important components of MT. However, defining MRs can be difficult. Chen et al. [8] proposed a specification-based method and developed a tool called MR-GENerator for identifying MRs based on category-choice framework [9]. Zhang et al. [10] proposed a search-based approach to automatic inference of polynomial MRs for a software under test, where a set of parameters is used to represent polynomial MRs, and the problem of inferring MRs is turn into a problem of searching for suitable values of the parameters. Then, particle swarm optimization is used to solve the search problem. Sun et al. [11] proposes a data-mutation directed metamorphic relation acquisition methodology, in which data mutation is employed to construct input relations and the generic mapping rule associated with each mutation operator to construct output relations. Liu et al. [12] proposed to systematically construct MRs based on some already identified MRs.
Without doubt, "good" MRs can improve the fault detection efficiency of MT. Chen et al. [13] reported that good MRs are those that can make the execution of the source-test case as different as possible to its follow-up test case. This perspective has been confirmed by the later studies [14], [15]. Asrafi et al. [16] conduct a case study to analyze the relationship between the execution behavior and the fault-detection effectiveness of metamorphic relations by code coverage criteria, and the results showed a strong correlation between the code coverage achieved by a metamorphic relation and its fault-detection effectiveness.
Source test cases also have a important impact on the fault detection effectiveness of MT. Chen et al. [17] compared the effects of source test cases generated by special value testing and random testing on the effectiveness of MT, and found that MT can be used as a complementary test method to special value testing. Batra and Sengupta [15] integrated genetic algorithms into MT to select source test cases maximising the the paths traversed in the software under test. Dong et al. [14] proposed a Path–Combination–Based MT method that first generates symbolic input for each executable paths and minis relationships among these symbolic inputs and their outputs, then constructs MRs on the basis of these relationships, and generates actual test cases corresponding to the symbolic inputs.
Different from the above investigates, we focused on performing test cases and MRs with fault revealing capabilities as quickly as possible by making use of feedback information. We first divided the input domain into disjoint partitions, and randomly selected an MR to generate follow-up test cases depended on source test case of related input partitions, then updated the test profile of input partitions according to the results of test execution. Next, a partition was selected according to updated test profile, and an MR was randomly selected from the set of MRs whose source test cases belong to selected partition.

2 *Combination with other techniques:* In order to improve the applicability and effectiveness of MT, it has been integrated into other techniques. Xie et al. [18] combined the MT with the spectrum-based fault localization (SBFL), extend the application of SBFL to the common situations where test oracles do not exist. Dong et al. [19] proposed a method for improving the efficiency of evolutionary testing (ET) by considering MR when fitness function is constructed. Liu et al. [20] introduced MT into fault tolerance and proposed a theoretical framework of a new technique called Metamorphic Fault Tolerance (MFT), which can handle system failure without the need of oracles during failure detection. In MFT, the trustworthiness of a test case 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 test case is.

3 *Application of MT:* Sun et al. [21], [22] proposed a metamorphic testing framework for web services taking into

account the unique features of SOA, in which MRs are derived from the description or Web Service Description Language (WSDL) [21] of the Web service, and on the basis on of MRs, follow-up test cses are generated depended on source test cases that are randomly generated according to the WSDL. Segura et al. [23] present a metamorphic testing approach for the detection of faults in RESTful Web APIs where they proposed six abstract relations called Metamorphic Relation Output Patterns (MROPs) that can then be instantiated into one or more concrete metamorphic relations. To evaluate this approach, they used both automatically seeded and real faults in six subject Web APIs.

V. Conclusion

In our future work, we plan to conduct experiments on more real-life programs to further validate the effectiveness of MT, and identify the limitations of our approach.

Acknowledgment

References

[1] E. J. Weyuker, "On testing non-testable programs," *The Computer Journal*, vol. 25, no. 4, pp. 465–470, 1982.

[2] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, 2015.

[3] K. Patel and R. M. Hierons, "A mapping study on testing non-testable systems," *Software Quality Journal*, vol. 26, no. 4, pp. 1373–1413, 2018.

[4] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: A new approach for generating next test cases," Department of Computer Science , Hong Kong University of Science and Technology, Hong Kong, Tech. Rep. HKUST-CS98-01, Tech. Rep., 1998.

[5] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. Tse, and Z. Q. Zhou, "Metamorphic testing: A review of challenges and opportunities," *ACM Computing Surveys*, vol. 51, no. 1, p. 4, 2018.

[6] M. Herlihy and N. Shavit, "Priority queues," in *The Art of Multiprocessor Programming*. Elsevier, 2012, ch. 15, pp. 351–368.

[7] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortés, "A survey on metamorphic testing," *IEEE Transactions on Software Engineering*, vol. 42, no. 9, pp. 805–824, 2016.

[8] T. Y. Chen, P.-L. Poon, and X. Xie, "Metric: Metamorphic relation identification based on the category-choice framework," *Journal of Systems and Software*, vol. 116, pp. 177–190, 2016.

[9] T. J. Ostrand and M. J. Balcer, "The category-partition method for specifying and generating fuctional tests," *Communications of the ACM*, vol. 31, no. 6, pp. 676–686, 1988.

[10] J. Zhang, J. Chen, D. Hao, Y. Xiong, B. Xie, L. Zhang, and H. Mei, "Search-based inference of polynomial metamorphic relations," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE'14)*, 2014, pp. 701–712.

[11] C.-A. Sun, Y. Liu, Z. Wang, and W. K. Chan, "μmt: A data mutation directed metamorphic relation acquisition methodology," in *Proceedings of the 1st International Workshop on Metamorphic Testing (MET'16), Co-located with the 38th International Conference on Software Engineering (ICSE'16)*, 2016, pp. 12–18.

[12] H. Liu, X. Liu, and T. Y. Chen, "A new method for constructing metamorphic relations," in *Proceedings of the 12th International Conference on Quality Software (QSIC'12)*, 2012, pp. 59–68.

[13] T. Y. Chen, D. Huang, T. Tse, and Z. Q. Zhou, "Case studies on the selection of useful relations in metamorphic testing," in *Proceedings of the 4th lberoAmerican Symposium on Software Engineering and Knowledge Engineer-ing (JIISIC'04)*, 2004, pp. 569–583.

[14] G. Dong, T. Guo, and P. Zhang, "Security assurance with program path analysis and metamorphic testing," in *Proceedings of the 4th IEEE International Conference on Software Engineering and Service Science (ICSESS'13)*, 2013, pp. 193–197.

[15] G. Batra and J. Sengupta, "An efficient metamorphic testing technique using genetic algorithm," in *International Conference on Information Intelligence, Systems, Technology and Management*, 2011, pp. 180–188.

[16] M. Asrafi, H. Liu, and F.-C. Kuo, "On testing effectiveness of metamorphic relations: A case study," in *Proceedings of the 15th International Conference on Secure Software Integration and Reliability Improvement (SSIRI'11)*, 2011, pp. 147–156.

[17] T. Y. Chen, F.-C. Kuo, Y. Liu, and A. Tang, "Metamorphic testing and testing with special values." in *Proceedings of the 5th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD'04)*, 2004, pp. 128–134.

[18] X. Xie, W. E. Wong, T. Y. Chen, and B. Xu, "Metamorphic slice: An application in spectrum-based fault localization," *Information and Software Technology*, vol. 55, no. 5, pp. 866–879, 2013.

[19] G. Dong, S. Wu, G. Wang, T. Guo, and Y. Huang, "Security assurance with metamorphic testing and genetic algorithm," in *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT'10)*, vol. 3, 2010, pp. 397–401.

[20] H. Liu, I. I. Yusuf, H. W. Schmidt, and T. Y. Chen, "Metamorphic fault tolerance: An automated and systematic methodology for fault tolerance in the absence of test oracle," in *Proceedings of the 36th International Conference on Software Engineering (ICSE'14)*, 2014, pp. 420–423.

[21] 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 the 9th IEEE International Conference on Web Services (ICWS'11)*, 2011, pp. 283–290.

[22] ——, "A metamorphic relation-based approach to testing web services without oracles," *International Journal of Web Services Research*, vol. 9, no. 1, pp. 51–73, 2012.

[23] S. Segura, J. A. Parejo, J. Troya, and A. Ruiz-Cortés, "Metamorphic testing of restful web apis," *IEEE Transactions on Software Engineering*, vol. 44, no. 11, pp. 1083–1099, 2018.