Theoretical and Empirical Analyses of the Effectiveness of Metamorphic Relation Composition

Kun Qiu, Zheng Zheng*, *Senior Member, IEEE,* Tsong Yueh Chen, *Member, IEEE,* and Pak-Lok Poon, *Member, IEEE*

Abstract—Metamorphic Relations (MRs) play a key role in determining the fault detection capability of Metamorphic Testing (MT). As human judgement is required for MR identification, systematic MR generation has long been an important research area in MT. Additionally, due to the extra program executions required for follow-up test cases, some concerns have been raised about MT cost-effectiveness. Consequently, the reduction in testing costs associated with MT has become another important issue to be addressed. MR composition can address both of these problems. This technique can automatically generate new MRs by composing existing ones, thereby reducing the number of follow-up test cases. Despite this advantage, previous studies on MR composition have empirically shown that some composite MRs have lower fault detection capability than their corresponding component MRs. To investigate this issue, we performed theoretical and empirical analyses to identify what characteristics component MRs should possess so that their corresponding composite MR has at least the same fault detection capability as the component MRs do. We have also derived a convenient, but effective guideline so that the fault detection capability of MT will most likely not be reduced after composition.

Index Terms—Metamorphic testing, metamorphic relation, metamorphic relation composition, test oracle, fault detection capability.

1 Introduction

Testing is a prominent technique for software verification [1], [2]. This technique often requires the presence of a *test oracle* (or simply an *oracle*, which refers to some mechanism for the tester to verify the correctness of the software output). However, in many situations such as testing a complex numerical algorithm, the "expected" correct software output (i.e., the oracle) is often unavailable or infeasible to determine. This problem is known as the *oracle problem*, which refers to the situation where either an oracle does not exist, or an oracle does exist but cannot be practically used, possibly due to resource constraints.

Some approaches or techniques have been proposed to alleviate the oracle problem in testing [3]. Among them, metamorphic testing (MT) has been demonstrated by various studies to be a lightweight, yet effective technique. When applying MT, the necessary properties of the software under test are firstly identified from various sources, such as the software specification. These properties are expressed in the form of relations among software inputs and outputs, formally known as *metamorphic relations* (*MRs*). Since its first publications in 1998 [4], [5], MT has been repeatedly found to be effective at alleviating the oracle problem in

* *Corresponding author.*

- *K. Qiu and Z. Zheng are with the School of of Automation Science and Electrical Engineering, Beihang University, Beijing 100191, China. E-mail: {qiukun, zhengz}@buaa.edu.cn.*
- *T. Y. Chen is with the Department of Computer Science and Software Engineering, Swinburne University of Technology, Hawthorn VIC 3122, Australia. E-mail: tychen@swin.edu.au.*
- *P.-L. Poon is with the School of Engineering and Technology, Central Queensland University, Melbourne VIC 3000, Australia. E-mail:frits*

composition. On the other hand, Liu et al. [31] reported that the fault detection capability may be reduced after MR composition, but without mentioning: (a) under what situations the fault detection capability will be reduced; or (b) how to avoid such a reduction happening.

In view of the above mixed findings, this paper aims to answer the question: ***In what situations is testing with a composite MR more cost-effective than testing with its component MRs?*** Our analysis has discovered some desirable characteristics in the component MRs, which can be easily verified by the tester, that indicate whether or not the composite MR should be used instead for testing. These characteristics are defined in terms of the bijectivity / injectivity of the input / output mappings of the component MRs. In addition to formally proving the validity of these characteristics, we have also checked their practicality through an empirical study. Furthermore, we propose a useful guideline for a tester to better decide whether or not to use MR composition. In brief, given a pair of metamorphic relations MR and MR that fall into a special class of MRs (as defined in Definition 1 of Section 3.1), where MR is composable with MR , they should be used to form a composite MR if both of the following conditions are met: (a) the output mapping of MR is injective; and (b) the input mapping of MR is a bijective mapping from the source inputs of MR to the source inputs of MR .

The rest of this paper is structured as follows. Section 2 outlines the concept of MT and gives the motivation for this study. Section 3 provides the basic concepts and terminology, which facilitate the subsequent discussion of our theoretical analysis of MR composition, which is discussed in detail in Section 4. Section 5 complements Section 4 by discussing our empirical analysis of MR composition. Based on the analyses in Sections 4 and 5 , Section 6 presents our general guideline for MR composition and an analysis of its applicability. This section also discusses some related works. Finally, Section 7 summarizes and concludes the paper.

2 BACKGROUND

2.1 Metamorphic Testing (MT)

MT is a lightweight, elegant, and effective technique for alleviating the oracle problem. An intuition underlying MT is as follows: Even if we cannot verify the correctness of an *individual* output, it may still be possible to use the relations among multiple inputs and outputs for program verification.

Example 1 (Shortest Path in an Undirected Graph): Consider an algorithm f for computing the length of the shortest path between any two nodes (a and b) in an undirected graph G. Let: (i) P denote an implementation of f; (ii) $P[G, a, b]$ denote the length of the shortest path between a and b in G, which is computed by P; and (iii) $f(G, a, b)$ denote the expected (and correct) length of the shortest path. When G contains many nodes and edges, for any a and b in G, it is resource-intensive and time-consuming to compute $f(G, a, b)$ in a brute force manner for comparing with $P[G, a, b]$. This is because the computational complexity is of factorial order of the number of nodes in G.

With MT, this tedious verification task can be alleviated by checking, for example, two properties, which are expressed as metamorphic relations MR_1 and MR_2, as follows:

- MR_1: If a and b in G are swapped, then $f(G, a, b) = f(G, b, a)$;
- MR_2: If G' is a permutation of G with a' and b' being the permuted counterparts of a and b, respectively, then $f(G', a', b') = f(G, a, b)$.

With respect to MR_1 and MR_2, we should have $P[G, a, b] = P[G, b, a]$ and $P[G, a, b] = P[G', a', b']$. Otherwise, we can conclude that P is faulty. □

Since its first publicaton in 1998 [4], [5], MT has been successfully applied across a wide range of application domains and platforms. Recently, funded by the UK Engineering and Physical Sciences Research Council and the TETRACOM (TEchnology TRAnsfer in COMputing systems) EU project, a group of academics and researchers from the Department of Computing at Imperial College London (ICL) established GraphicsFuzz — a spinout company from ICL. GraphicsFuzz [16] combined fuzzing and MT to test graphics drivers. The company was acquired by Google in 2018.

A core concept of MT is the MR, which is a necessary property of a targeted function f. An MR of f is a relation over a sequence of two or more inputs $ht_1, t_2, ..., t\ \ i$ and their corresponding outputs $hf(t_1), f(t_2), ..., f(t\ \ )i$, where $n \quad 2$. An MR can be written as $\mathbb{R} \quad X \quad Y$, where $X \quad Y$ are Cartesian products of the n inputs and their corresponding n outputs. Generally, an MR can be represented as:

$$\mathbb{R}(t_1,\ t_2,\ ...,\ t\ \ ,\ f(t_1),\ f(t_2),\ \ldots,\ f(t\ \ )).$$

Consider, for instance, MR_1 in Example 1, which can be rewritten as:

$$\mathbb{R}((G, a, b), (G, b, a), f(G, a, b), f(G, b, a)).$$

Give any MR, there exists a k, where $1 \quad k < n$, such that:

- $t_1,\ t_2,\ \ldots,\ t\ \ $ denote the *source inputs*;
- $f(t_1),\ f(t_2),\ \ldots,\ f(t\ \ )$ denote the *source outputs*;
- $t\ _{+1},\ t\ _{+2},\ \ldots,\ t\ \ $ denote the *follow-up inputs*;
- $f(t\ _{+1}),\ f(t\ _{+2}),\ \ldots,\ f(t\ \ )$ denote the *follow-up outputs*.

Let P be an implementation of f. With respect to an MR, applying MT typically proceeds as follows:

(1) Replace f with P in $\mathbb{R}$.
(2) Execute P on a sequence of source inputs $ht_1,\ t_2,\ \ldots,\ t\ \ i$ to obtain the corresponding sequence of source outputs $hP[t_1],\ P[t_2],\ \ldots,\ P[t\ \ ]i$.
(3) Generate a sequence of follow-up inputs $ht\ _{+1},\ t\ _{+2},\ \ldots,\ t\ \ i$ in accordance with $\mathbb{R}$.
(4) Execute P on the sequence of follow-up inputs to obtain the corresponding sequence of outputs $hP[t\ _{+1}],\ P[t\ _{+2}],\ \ldots,\ P[t\ \ ]i$.
(5) Compare the two sets of execution results with reference to $\mathbb{R}$. If $\mathbb{R}$ is violated, then P is faulty.

The above steps are repeated for every identified MR.

2.2 MR Composition

The composition technique was originally proposed as a method of generating new MRs from existing ones [30], [31]. Example 2 explains the basic concept of MR composition.

Example 2 (MR Composition): Consider the following two MRs, corresponding to two well-known properties of the $sine$ function:

- MR_1: If $x' = \quad x$, then $sin(x') = \quad sin(x)$;
- MR_2: If $x' = x + 2\pi$, then $sin(x') = sin(x)$.

We can compose MR_1 and MR_2 together to form a composite metamorphic relation MR_{12}, that is $MR_1(MR_2)$. MR_{12} is formally expressed as: If $x' = \quad (x + 2\pi)$, then $sin(x') = \quad sin(x)$. □

Example 2 above shows that MR composition can generate new MRs from existing ones. If we only generate test cases from MR_{12} (and ignore MR_1 and MR_2) for testing, the testing cost is obviously reduced. In Example 2, testing with both MR_1 and MR_2 involves three program executions: one for $sin(x)$, one for $sin(x + 2\pi)$, and one for $sin(\quad x)$. On the other hand, testing with MR_{12} only involves two executions: one for $sin(x)$ and another for $sin(\quad (x + 2\pi))$. Thus, if we only consider the testing cost, testing with MR_{12} alone is definitely preferable to testing with both MR_1 and MR_2. However, beyond savings in testing costs, we should also compare the fault detection capability of MR_{12} with that of MR_1 and MR_2. This leads to the research question of this paper: ***(RQ) Will testing a composite MR (e.g., MR_{12}) have the same chance of detecting faults when compared with testing its component MRs (e.g., MR_1 and MR_2)?***

Previous studies on MR composition do not provide a definite answer to RQ. For instance, the case study reported by Dong et al. [30] has provided a "Yes" answer to RQ. On the other hand, the study by Liu et al. [31] reported that this is not necessarily the case. To date, to the best of our knowledge, no systematic studies have been conducted to address our RQ. In view of this, we perform a theoretical analysis with the intention of providing a definite answer.

3 Important Concepts and Terminology

Before we present our theoretical analysis, we first formalize some important definitions and concepts.

3.1 Metamorphic Relations (MRs)

In this paper, without loss of generality, we assume that the targeted function (or algorithm) involves one single input and one single output. However, generalizing our results to functions with multiple inputs and outputs is straightforward.

Because composing any two MRs into their corresponding composite MR is not always feasible, our work only considers the special class of MRs defined in this subsection. Before formally presenting this special class of MRs, let us revisit some basic concepts of mapping.

Basic Concepts of a Function

Let $f\colon A \ ! \ B$ be a function (or mapping) from A to B. Here:

- A is referred to as the *domain* of f;
- B is referred to as the *codomain* of f;
- If $f(a) = b$, then b is referred to as the *image* of a, and a is referred to as the *preimage* of b;
- f is said to be *injective* if $8\,a,\ a'\ 2\ A,\ f(a) = f(a')$ implies $a = a'$;
- f is said to be *surjective* if $8\,b\ 2\ B,\ 9\,a\ 2\ A$ such that $f(a) = b$;
- f is said to be *bijective* if f is both injective and surjective;
- For any $S \quad A$, $f(S)$ denotes the set of the images of elements in S under the function f, such that:

$$f(S) = \bigcup_{\in S} \mathit{f}\, f\,(s)\mathit{g};$$

- $f(A)$ is referred to as the *range* of f, and $f(A) \quad B$.

The specific class of MRs considered in our study is defined as follows:

Definition 1. A Special Class of Metamorphic Relations (MRs)

Let

- $f\colon T \ ! \ R$ be a targeted function;
- $I\colon T \ ! \ T'$ (where $T \quad T$, and $T' = I(T) \quad T$) be a mapping that takes in a source input and generates a follow-up input for f;
- $O\colon R \ ! \ R'$ (where $R = f(T)$ and $R' = O(R)$ R) be a mapping that takes in a source output (i.e., $f(t)$) and generates a follow-up output.

A metamorphic relation MR is a necessary property of f. MR is

t denote an input of f. Then, for any $t \in T$, t is of the form of $\langle G, a, b \rangle$. Additionally,

- MR_1: $\forall t \in T_1 (f(I_1(t)) = O_1(f(t)))$, where $T_1 = T$, $I_1(\langle G, a, b \rangle) = \langle G, b, a \rangle$, and $O_1(x) = x$.
- MR_2: $\forall t \in T_2 (f(I_2(t)) = O_2(f(t)))$, where $T_2 = T$, $I_2(\,)$ is a permutation function that takes in $\langle G, a, b \rangle$ and permutates G, a, and b according to a certain pattern, such that $I_2(\langle G, a, b \rangle) = \langle G', a', b' \rangle$, where G' is the permuted G, a' and b' are the counterparts of a and b, respectively, and $O_2(x) = x$.

Next, consider Example 2. Let T and R be the domain and codomain of the $sine$ function, respectively; we write $sine$ as f and t as the input of f. Then:

- MR_3: $\forall t \in T_3 (f(I_3(t)) = O_3(f(t)))$, where $T_3 = T$, $I_3(t) = t$, and $O_3(x) = x$;
- MR_4: $\forall t \in T_4 (f(I_4(t)) = O_4(f(t)))$, where $T_4 = T$, $I_4(t) = t + 2\pi$, and $O_4(x) = x$.

3.2 Composable MR and Composite MR

Below we first define a composable MR, which will facilitate the definition of a composite MR.

Definition 2. Composable MR

Let

- $f : T \to R$ be a targeted function;
- MR_a and MR_b be two MRs of f.

MR_a is said to be *composable* with MR_b if:

- $I_a(T_a) \subseteq T_b$, that is, the range of I_a is a subset of the domain of I_b;
- $O_a(R_a) \subseteq R_b$ (where $R_a = f(T_a)$ and $R_b = f(T_b)$), that is, the range of O_a is a subset of the domain of O_b.

For the rest of the paper, we use subscripts to link an MR and its related components. For instance, in Definition 2 above, T_a, I_a, and O_a denote the set of source inputs, input mapping, and output mapping corresponding to MR_a, respectively.

Refer to Example 1. It can be deduced that:

(a) $I_1(T_1) = T_1 = T_2$ because (i) $I_1(T_1) = T_1$ and (ii) $T_1 = T_2 = T$;
(b) $O_1(R_1) = R_1 = R_2$ because (i) $R_1 = R_2$ as $f(T_1) = f(T_2)$ and (ii) $O_1(R_1) = R_1$ because $O_1(x) = x$;
(c) $I_2(T_2) = T_2 = T_1$ because (i) $I_2(T_2) = T_2$ as I_2 is a permutation function and (ii) $T_1 = T_2 = T$;
(d) $O_2(R_2) = R_2 = R_1$ because (i) $R_2 = R_1$ as $f(T_2) = f(T_1)$ and (ii) $O_2(R_2) = R_2$ as we have $O_2(x) = x$.

With (a) and (b), it follows after Definition 2 that MR_2 is composable with MR_1. Similarly, with (c) and (d), we conclude that MR_1 is composable with MR_2. It should, however, be noted that for any two metamorphic relations MR_a and MR_b, if MR_a is composable with MR_b, it is not necessary that MR_b is also composable with MR_a.

We next formally define the construction of a composite MR:

Definition 3. Composite MR and Component MR

Let

- $f : T \to R$ be a targeted function;
- MR_a and MR_b be two MRs of f;
- MR_a be composable with MR_b.

The *composite MR* (denoted by MR_c), formed by composing MR_a with MR_b, is a necessary property of f. MR_c is formally expressed as follows:

$$\forall t \in T_a \quad (f(I_c(t)) = O_c(f(t))),$$

where

- $T_c = T_a$;
- $I_c(t) = (I_b \circ I_a)(t) = I_b(I_a(t))$;
- $O_c(f(t)) = (O_b \circ O_a)(f(t)) = O_b(O_a(f(t)))$.

We write $MR_c = MR_a \circ MR_b$ or $MR_c(MR_a \circ MR_b)$. Also, we refer to MR_a and MR_b as the *component metamorphic relations* of MR_c.

Refer to Example 2. It is straightforward to conclude that MR_1 and MR_2 are composable with each other according to Definition 2. We write $f(t)$ as $sin(t)$. Then, with respect to MR_1 and MR_2, there are two possible composite MRs.

- $MR_{12} = MR_1(MR_2) = MR_1 \circ MR_2$:

  $$\forall t \in T_{12} (f(I_{12}(t)) = O_{12}(f(t))),$$

  where $T_{12} = T_2 = T$, $I_{12}(t) = I_1(I_2(t)) = (t + 2\pi)$ and $O_{12}(x) = O_1(O_2(x)) = x$.

- $MR_{21} = MR_2(MR_1) = MR_2 \circ MR_1$:

  $$\forall t \in T_{21} (f(I_{21}(t)) = O_{21}(f(t))),$$

  where $T_{21} = T_1 = T$, $I_{21}(t) = I_2(I_1(t)) = t + 2\pi$, and $O_{21}(x) = O_2(O_1(x)) = x$.

It should be noted that, when applying MT, the tester is not required to explicitly specify the composite MR in the format above: The task of composing composable MRs can be automated through programming in accordance with Definition 3. This automation can be implemented as follows. Two test scripts can be written — one calling function I_a and the other calling function I_b. If t is an input to I_b, then the returned value from I_a is used as an input to I_b. In this way, t and the returned value from I_a form a pair of source and follow-up inputs for the composite MR_c. Similarly, the pair of source and follow-up outputs for MR_c could be obtained by first executing a test script to call the function O_a, followed by executing another test script to call the function O_b. Here, the functions I_a, I_b, O_a, and O_b are implemented according to MR_a and MR_b.

When composing more than two MRs, the resultant composite MR can also be automatically obtained by recursively applying Definition 3. Since I and O are mappings, the composition of Is and the composition of Os are associative, that is: $I_a \circ (I_b \circ I_c) = (I_a \circ I_b) \circ I_c = I_a \circ I_b \circ I_c$ and $O_a \circ (O_b \circ O_c) = (O_a \circ O_b) \circ O_c = O_a \circ O_b \circ O_c$. Since an MR is defined in terms of its own I and O, therefore, the composition of MRs is also associative. For example, $MR_a \circ (MR_b \circ MR_c) = (MR_a \circ MR_b) \circ MR_c = MR_a \circ MR_b \circ MR_c$. The

order of composition, however, is important. For instance, MR may not be the same as MR . In other words, the composition of MRs is not commutative.

3.3 Evaluation of Fault Detection Capability

The evaluation and comparison of the fault detection capabilities of two different MRs (regardless of whether or not they are composite) requires some metric. We defined two — one qualitative, and one quantitative — to use in our study. To facilitate the definition of these two metrics, we first present the following definition:

Definition 4. Satisfiability of a Set of Source Inputs for an MR

Let

- f be a targeted function;
- P be an implementation of f;
- MR be a metamorphic relation of f with T, I, and O being its set of source inputs, input mapping, and output mapping, respectively;
- S be a nonempty subset of T.

After executing all elements of S with P,

- S is said to *satisfy* MR, if all elements of S satisfy MR, that is,

$$8t 2 S\left(O\left(P\left[t\right]\right) = P\left[I\left(t\right)\right]\right);$$

- S is said to *violate* MR, if all elements of S violate MR, that is,

$$8t 2 S\left(O\left(P\left[t\right]\right) 6= P\left[I\left(t\right)\right]\right).$$

With Definition 4, a qualitative metric is defined for measuring the fault detection capability of an MR as follows:

Definition 5. Satisfiability of an MR

Let

- f be a targeted function;
- P be an implementation of f;
- MR be a metamorphic relation of f with T being its set of source inputs.

With respect to P, if T satisfies MR, MR is said to be *satisfiable*; otherwise, MR is said to be *violative*.

Let T denote the set of all elements in T that violate MR. In this case, T is referred to as the *set of violative source inputs* of MR. Obviously, $T = ;$, iff MR is *satisfiable*.

In Definition 5 above, given an implementation P and an MR, it is equivalent to say that P violates (or satisfies) the MR, when the MR is violative (or satisfiable).

Together Definitions 4 and 5 allow us to define the following quantitative metric for measuring the fault detection capability of an MR:

Definition 6. Fault Detection Rate of an MR

Let

- f be a targeted function;
- P be an implementation of f;
- MR be a metamorphic relation of f with T being its set of source inputs and T being its set of violative source inputs.

Let θ denote the *fault detection rate* of MR with respect to P. Then, θ is defined as follows:

$$\theta = \frac{jT\ j}{jTj},$$

where $jT\ j$ and jTj denote the sizes of T and T, respectively.

The satisfiability of an MR (Definition 5) indicates whether or not a program under test can be revealed as faulty by this MR. Furthermore, the fault detection rate (Definition 6) indicates *how likely* it is that an MR will reveal a fault in P with only one *random* source input. Larger fault detection rates indicate higher fault detection capabilities.

4 Theoretical Analysis of Fault Detection Capability

Given an implementation P, MR , and MR , there are four possible scenarios:

(1) Both MR and MR are satisfiable;
(2) MR is satisfiable and MR is violative;
(3) MR is violative and MR is satisfiable;
(4) Both MR and MR are violative.

For each of the above scenarios, we analyze the fault detection capability of MR .

4.1 Scenario 1

In this scenario, both MR and MR are satisfiable, that is, $\theta = \theta = 0$. Although we intuitively expect θ to be 0, let us formally prove it (Theorem 1). This proof needs the following lemma.

Lemma 1.

Let

- f be a targeted function;
- P be an implementation of f;
- MR and MR be two MRs of f;
- MR be composable with MR ;
- S be a nonempty subset of T .

If S satisfies MR and $I\ (S\ )$ satisfies MR , then S satisfies MR .

Proof (Lemma 1). Assume that S satisfies MR and $I\ (S\ )$ satisfies MR . It follows after Definition 4 that

$$8t 2 S\ \ (O\ \ (P\left[t\right]) = P\left[I\ \ (t)\right]), \quad (1)$$

and

$$\forall t' \in I(S)(O(P[t']) = P[I(t')]). \quad (2)$$

By the definition of $I(S)$, for any $t' \in I(S)$, there exists a $t \in S$ such that $t' = I(t)$; and for any $t \in S$, there exists a $t' \in I(S)$ such that $t' = I(t)$. Therefore, Eq. 2 can be rewritten as follows:

$$\forall t \in S(O(P[I(t)]) = P[I(I(t))]). \quad (3)$$

Immediately after Eqs. 1 and 3, we have

$$\forall t \in S(O(O(P[t])) = P[I(I(t))]). \quad (4)$$

It follows after Definition 4 that S satisfies MR_2. □

Now we are ready to present Theorem 1 and its proof.

Theorem 1.

Let

- f *be a targeted function;*
- P *be an implementation of* f*;*
- MR_1 *and* MR_2 *be two MRs of* f*;*
- MR_1 *be composable with* MR_2*.*

If MR_1 *and* MR_2 *are satisfiable, then their composite metamorphic relation,* MR_3*, is also satisfiable.*

Proof (Theorem 1). Assume that MR_1 and MR_2 are both satisfiable. From Definition 5, we immediately have T satisfies MR_1 and T satisfies MR_2. Since MR_1 is composable with MR_2, it follows after Definition 2 that $I(T) \subseteq T$. Since T satisfies MR_2, $I(T)$ also satisfies MR_2.

It follows from Lemma 1 that, since T satisfies MR_1 and $I(T)$ satisfies MR_2, T satisfies MR_3. Because $T = T$ (Definition 3), therefore, it follows after Definition 5 that MR_3 is satisfiable, that is, $\theta = 0$. □

Implication: Theorem 1 states that, if an implementation P under test does not violate any two component MRs (MR_1 and MR_2), P will also not violate any composite MR constructed from MR_1 and MR_2.

4.2 Scenario 2

In this scenario, MR_1 is satisfiable ($\theta = 0$) and MR_2 is violative ($\theta > 0$). Before analyzing the fault detection capability of MR_3, we first introduce the following lemma to facilitate the proof of Theorem 2.

Lemma 2.

Let

- f be a targeted function;
- P be an implementation of f;
- MR_1 and MR_2 be two MRs of f;
- MR_1 be composable with MR_2;
- S be a nonempty subset of T.

Suppose that S violates MR_1 and $I(S)$ satisfies MR_2. If O is an injective mapping, then S violates MR_3.

Proof (Lemma 2). Since S violates MR_1, it follows after Definition 4 that

$$\forall t \in S(O(P[t]) \neq P[I(t)]). \quad (5)$$

Because $I(S)$ satisfies MR_2, it follows after Definition 4 that

$$\forall t' \in I(S)(O(P[t']) = P[I(t')]). \quad (6)$$

By the definition of $I(S)$, for any $t' \in I(S)$, there exists a $t \in S$ such that $t' = I(t)$; and for any $t \in S$, there exists a $t' \in I(S)$ such that $t' = I(t)$. Therefore, Eq. 6 can be rewritten as follows:

$$\forall t \in S(O(P[I(t)]) = P[I(I(t))]). \quad (7)$$

Assume that O is an injective mapping. Immediately after Eqs. 5 and 7, we have

$$\forall t \in S(O(O(P[t])) \neq P[I(I(t))]). \quad (8)$$

Therefore, S violates MR_3. □

With Lemma 2, we can now introduce Theorem 2 and its proof.

Theorem 2.

Let

- f *be a targeted function;*
- P *be an implementation of* f*;*
- MR_1 *and* MR_2 *be two MRs of* f*;*
- MR_1 *be composable with* MR_2*.*

Suppose that MR_1 *is satisfiable (*$\theta = 0$*) and* MR_2 *is violative (*$\theta > 0$*). If* O *is an injective mapping, then* MR_3 *is violative and* $\theta = \theta$*.*

Proof (Theorem 2). To determine θ, we need to know the set of source inputs (T) and the set of violative source inputs (T) of MR_3. It follows from Definition 3 that $T = T$. In what follows, we will prove that, if O is an injective mapping, then $T = T$.

Since MR_1 is violative, we have $T = T \cup \overline{T}$[1], where $T \neq \emptyset$. Since MR_1 is composable with MR_2, we have $I(T) \subseteq T$. In turn, we have $I(T) \subseteq T$ and $I(\overline{T}) \subseteq T$. Since MR_2 is satisfiable, it follows after Definition 5 that T satisfies MR_2. Therefore, we have

$$I(T) \text{ satisfies } MR_2; \quad \text{(i)}$$

and

$$I(\overline{T}) \text{ satisfies } MR_2, \text{ if } I(\overline{T}) \neq \emptyset. \quad \text{(ii)}$$

Next, let us assume that O is injective. Since we have (i) above and T violates MR_3, it follows from Lemma 2 that

1. In this paper, we use $\overline{T^v}$ to denote the complementary set of T^v over T. For instance, $\overline{T_y^v}$ is the complementary set of T_y^v over T_y.

T violates MR . (iii)

Furthermore, we have

$T \quad T$, (iv)

because T contains all the elements that violate MR .

Next, we consider the following two exhaustive cases:

Case (a): $\overline{T}$ $\neq$;.

Since we have (ii) and $\overline{T}$ satisfies MR , it follows from Lemma 1 that $\overline{T}$ satisfies MR . Furthermore, since we have (iii) and $T = T = T [\overline{T}$, therefore, we have $T = T$.

Case (b): $\overline{T} = ;$.

Since we have (iv), $T \quad T$ (where $T = T$), and $T = T [\overline{T} = T$, therefore, we have $T = T$.

In view of the above two cases, regardless of whether or not $\overline{T}$ is empty, we have $T = T$. Then, it follows after Definition 6 that,

$$\theta = \frac{jT \ j}{jT \ j} = \frac{jT \ j}{jT \ j} = \theta > 0.$$

In other words, MR is violative. □

Implication: Theorem 2 gives a sufficient condition for MR and MR having the same fault detection capability if MR is satisfiable.

A previous study [31] reported that composing some "loose" MRs may result in a composite MR with a lower fault detection capability. However, that study has not formally defined the meaning of "loose" MRs. By means of our theoretical analysis, we found that a "loose" MR in fact refers to one whose output mapping is not injective.

4.3 Scenario 3

In this scenario, MR is violative ($\theta > 0$) and MR is satisfiable ($\theta = 0$). Before introducing Theorems 3 and 4, we need the following lemma to facilitate their proofs.

Lemma 3.

Let

- f be a targeted function;
- P be an implementation of f;
- MR and MR be two MRs of f;
- MR be composable with MR ;
- S be a nonempty subset of T .

If S satisfies MR and $I(S)$ violates MR , then S violates MR .

Proof (Lemma 3). Assume that there exists a nonempty S T such that S satisfies MR and $I(S)$ violates MR . Since S satisfies MR , it follows after Definition 4 that

$$8t \ 2 \ S \ (O \ (P[t]) = P[I \ (t)]). \quad (9)$$

Because $I(S)$ violates MR , it follows after Definition 4 that

$$8t' \ 2 \ I \ (S \ )(O \ (P[t']) \ 6= P[I \ (t')]). \quad (10)$$

By the definition of $I(S)$, for any $t' \ 2 \ I \ (S \ )$, there exists a $t \ 2 \ S$ such that $t' = I(t)$; and for any $t \ 2 \ S$, there exists a $t' \ 2 \ I \ (S \ )$ such that $t' = I(t)$. Therefore, Eq. 10 can be rewritten as follows:

$$8t \ 2 \ S \ (O \ (P[I \ (t)]) \ 6= P[I \ (I \ (t))]). \quad (11)$$

Immediately after Eqs. 9 and 11 , we have

$$8t \ 2 \ S \ (O \ (O \ (P[t])) \ 6= P[I \ (I \ (t))]). \quad (12)$$

Therefore, S violates MR . □

Theorem 3.

Let

- f *be a targeted function;*
- P *be an implementation of* f*;*
- MR *and* MR *be two MRs of* f*;*
- MR *be composable with* MR *.*

Suppose that MR *is violative* ($\theta > 0$) *and* MR *is satisfiable* ($\theta = 0$). *If* $I(T) = T$, *then* MR *is violative with* $\theta = \frac{|a|}{|y|} > 0$ *(where* $T \ 6= ;$, $T \quad T$, *and* $I(T) = T$ *).*

Proof (Theorem 3). To determine θ , we need to know the set of source inputs (T) and the set of violative source inputs (T) of MR . It follows from Definition 3 that $T = T$. In what follows, we will prove that, if $I(T) = T$, then $T = T$ (where $T \ 6= ;$, $T \quad T$, and $I(T) = T$).

Since MR is violative, we have $T = T [\overline{T}$, where $T \ 6= ;$. Let us assume that: (a) $I(T) = T$; and (b) T , $T \quad T$, such that $I(T) = T$ and $I(T) = \overline{T}$. Since $T \ 6= ;$, we have $T \ 6= ;$ I , and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2020.3009698, IEEE Transactions on Software Engineering

TABLE 1: Fault Detection Rates (θ) of MR in Different Testing Scenarios

Scenario	θ_x	θ_y	$max\{\theta_x, \theta_y\}$	θ_{xy}
1	$= 0$	$= 0$	$= 0$	$\theta_{xy} = 0$ (Theorem 1)
2	$= 0$	> 0	$= \theta_y$	If O_x is injective, then $\theta_{xy} = \theta_y$ (Theorem 2)
3	> 0	$= 0$	$= \theta_x$	If $I_y(T_y) = T_x$, then $\theta_{xy} > 0$ (Theorem 3)
				If $I_y(T_y) = T_x$ and I_y is bijective, then $\theta_{xy} = \theta_x$ (Theorem 4)
4	> 0	> 0	Varies in different situations	If $I_y(T_y) = T_x$ and O_x is injective, then it is very likely to have $\theta_{xy} > 0$

$$A_3 \quad T \quad . \tag{viii}$$

Let us assume that MR is satisfiable, that is, $T = ;$ or $\theta = 0$. Immediately, with (vii) and (viii) above, we have $A_2 = A_3 = ;$. Because $T = A_1 \; [\; A_2 = A_1 = T \; \setminus \overline{T}$, we have $T \quad \overline{T}$. Since $T = A_3 \; [\; A_4 = A_4 = T \; \setminus T$, we have $T \quad T$. Furthermore, since $T = T \; [\; T = T \; [\; \overline{T}$, we have $T = T$ and $T = \overline{T}$. Since $I(T) = T$ and $I(T) = \overline{T}$ by definition, we have

$$I(T) = T, \tag{ix}$$

and

$$I(\overline{T}) = \overline{T}. \tag{x}$$

Since (ix) and (x) follow after the assumption that $\theta = 0$, they are necessary relations for θ to be 0. Relation (ix) implies that, for every violating source test input for MR, its corresponding follow-up input must be a violating source input for MR. Similarly, relation (x) implies that, for every non-violating source input for MR, its corresponding follow-up input must be a non-violating source input for MR. Obviously, these two relations are very tight and restrictive and, hence, they are unlikely to be satisfied simultaneously. Since the situation of $\theta = 0$ requires the simultaneous satisfaction of relations (ix) and (x) (which is very rare, as explained above), it can be comfortably concluded that the situation of $\theta > 0$ is very likely to occur. In summary, the above theoretical analysis has showed that our hypothesis will be strongly held because of two very tight and restrictive relations (ix) and (x). We performed an empirical study to support the above theoretical analysis for Scenario 4. Details will be given in Section 5.

Table 1 summarizes our theoretical analysis on the fault detection rates of MR in the four different testing scenarios.

5 Empirical Analysis of Fault Detection Capability

In the first three testing scenarios (Scenarios 1, 2, and 3) discussed in Sections 4.1, 4.2, and 4.3, we are able to obtain a definite answer after a theoretical analysis. In other words, for each of those three scenarios, we have found the characteristics that component MRs should possess to guarantee that a composite MR has the same chance of detecting the faults as its component MRs do. On the other hand, Scenario 4 is too complicated to have a definite answer solely based on a theoretical analysis. Nevertheless, our theoretical analysis of Scenario 4 has led to a hypothesis, as stated at the beginning of Section 4.4, which was further verified by an empirical study to be discussed in this section. We next discuss the settings and observations of our empirical study.

5.1 Subject Programs

Our empirical study involved the following four subject programs:

- *TriangleSquare (TSQ)*. It accepts three numbers, corresponding to the three edges of a triangle, and calculates its area if a legitimate triangle can be formed [30].
- *SparseMatrixMultiplication (SMM)*. It accepts two sparse matrices[2] as inputs and computes their product matrix [30].
- *Dnapars (DNA)*. It is commonly used in bioinformatics [8]. It takes in a matrix containing a set of species' DNAs and generates an evolution tree.
- *K-Nearest Neighbors (KNN)*. It is a machine learning classifier algorithm, which takes in a training data set and a testing data, and then predicts the label for the latter based on the former [19].

Table 2 gives more details about these subject programs, in terms of their inputs, outputs, and the approaches we used to assert two equivalent outputs.

5.2 Experimental Procedures

We applied the following steps to each subject program:

(1) For each identified MR, manually check its compliance with Definition 1. This check is required because Definition 1 specifies a special (but common) class of MRs that our study has assumed.
(2) For any tuple of two MRs $(MR \; , MR \;)$, use Definition 2 to manually check whether MR is composable with MR. If yes, then check whether: (a) O is injective; and (b) $I(T) = T$. If yes to both (a) and (b), then generate the composite MR (MR) from MR and MR.
(3) Apply mutation analysis and random testing to estimate individual fault detection rates for MR, MR, and MR.

5.3 Component and Composite MRs

Table 3 lists all the MRs used in our study; they were sourced from previous MT-related studies [8], [19], [30]. All these MRs were then checked and confirmed to comply with Definition 1 (see step 1). There were 23 $(7 + 7 + 6 + 3)$ such MRs for the four subject programs. The 3rd column of this

2. A sparse matrix is a matrix in which most of the elements are zero.

TABLE 2: Inputs, Outputs, and Equivalent Output Assertions of Subject Programs

Subject Programs	Inputs ( )	Outputs (( ))	Assertion of Equivalent Outputs
TSQ	$= \langle a, b, c \rangle$, where a, b, and c denote the three edges of a triangle	If the three edges form a legitimate triangle, then ( ) = , where denotes the area of the triangle	Let and ′ denote two outputs. Then, = ′ iff | − ′| < $\epsilon^{\dagger}$.
SMM	= $\langle A, B \rangle$, where A and B denote two sparse matrices in the Compressed Sparse Row (CSR) format	( ) = C, where C denotes the product of A and B	Let $C = [\ _{ij}]$ and $C' = [\ '_{ij}]$ denote two outputs. Then, $C = C'$ iff for all $\|\ '_{ij} - \ '_{ij}\| < \epsilon^{\dagger}$.
DNA	= X, where X denotes the species' DNAs and is in the format of a (×) matrix (= number of species; = length of a DNA sequence)	( ) = ⟨ , ⟩, where and denote the length and the structural description of the generated evolution tree, respectively	Let ⟨ , ⟩ and ⟨ ′, ′⟩ denote two outputs. Then, ⟨ , ⟩ = ⟨ ′, ′⟩ iff | − ′| < $\epsilon^{\dagger}$ and is identical to ′.
KNN	= $\langle X, C, S \rangle$, where X denotes the attributes of the training data set in the format of a (×) matrix (= number of entries of the training data set; = number of attributes of each entry), C denotes the class labels of the training data with a size of , and S denotes the testing item's attributes with a size of elements	( ) = c, where c denotes the calculated class label for S	Let c and c' denote two class labels. Then, $c = c'$ iff c and c' are exactly the same.

(†) ϵ is an extremely small value and is set to 10^{-} .

table gives the details on these MRs. For each such MR, we also explicitly list its set of source inputs (4th column), input mapping (5th column), and output mapping (6th column) in the table.

For any tuple of two MRs (MR , MR), checks were performed to ensure that conditions (a) and (b) in step 2 were fulfilled. After checking for all the tuples of two MRs, a total of 106 (*TSQ*=42, *SMM*=42, *DNA*=16, and *KNN*=6) eligible pairs of component MRs were found, thereby resulting in the construction of 106 composite MRs. As an example, let us consider MR_1 and MR_2 of *TSQ* in Table 3. We write the function corresponding to *TSQ* as f. It can be deduced that: (a) $T_2 = I_1(T_1) = T_1 = I_2(T_2) = T$ because both I_1 and I_2 are bijective mappings from T to T; and (b) $R_2 = O_1(R_1) = R_1 = O_2(R_2) = f(T)$ because both O_1 and O_2 are bijective mappings from $f(T)$ to $f(T)$. It then follows from Definition 2 that MR_1 and MR_2 are composable with each other. Therefore, MR_{12} and MR_{21} were formed and tested in our study.

5.4 Measurement of Fault Detection Rates and Generation of Mutants

According to Definition 6, the fault detection rate of an MR with respect to a program P is the ratio of $jT\ j$ (the size of the MR's set of violative source inputs) to jTj (the size of the MR's set of source inputs). However, because the size of T is often very large, therefore it is practically infeasible to conduct exhaustive testing to determine the value of T . In turn, we cannot compute an MR's fault detection rate based on Definition 6. Therefore, in this study, we used the following equation as the "estimator" of an MR's fault detection rate (θ):

$$\theta \ ' \ \frac{N}{N},$$

where N denotes the number of tested source inputs that caused violations to a given MR, and N denotes the total number of source inputs used in testing.

Mutation analysis [32] has long been used in MT to evaluate the fault detection rate of MRs (e.g., in [19]). Thus, we also used the mutation technique (together with random testing) to estimate the fault detection rates of component and composite MRs. Table 4 shows the mutants of the four subject programs with injected faults. We randomly generated $10\,000$ ($N = 10\,000$) source inputs for each of the four subject programs.

For each mutant of every subject program, we then performed two operations: (a) used each identified component MR and each constructed composite MR to generate a separate set of follow-up inputs from the set of source inputs (with a size of $10\,000$); and (b) executed these source inputs and follow-up inputs with the mutants and checked for violations to MRs.

5.5 Experimental Observations

Table 5 shows the estimated fault detection rates of component and composite MRs when $\theta \ > 0$ and $\theta \ > 0$ (a total of 108 such cases). The complete set of experimental data of the estimated fault detection rates for the four subject programs are given in Tables 7, 8, and 9 in the Appendix (as a separate file).

For ease3 pera an the2(able)-297(an)575(.)],97(an)Fig467(As)197(an)ws an c7lue an

TABLE 3: List of MRs Used in the Empirical Study

Sub. Pro.	MRs	Description	T	Input Mapping	Output Mapping
	MR_1	If $\langle a', b', c' \rangle = \langle b, a, c \rangle$, then $' =$	$T_1 = \mathcal{T}$	$I_1(\langle a, b, c \rangle) = \langle b, a, c \rangle$	$O_1(\ ) =$
	MR_2	If $\langle a', b', c' \rangle = \langle a, c, b \rangle$, then $' =$	$T_2 = \mathcal{T}$	$I_2(\langle a, b, c \rangle) = \langle a, c, b \rangle$	$O_2(\ ) =$
	MR_3	If $\langle a', b', c' \rangle = \langle c, b, a \rangle$, then $' =$	$T_3 = \mathcal{T}$	$I_3(\langle a, b, c \rangle) = \langle c, b, a \rangle$	$O_3(\ ) =$
TSQ	MR_4	If $\langle a', b', c' \rangle = \langle 2a, 2b, 2c \rangle$, then $' = 4$	$T_4 = \mathcal{T}$	$I_4(\langle a, b, c \rangle) = \langle 2a, 2b, 2c \rangle$	$O_4(\ ) = 4$
	MR	If $\langle a', b', c' \rangle = \langle \sqrt{2b^2 + 2c^2 - a^2}, b, c \rangle$, then $' =$	$T = \mathcal{T}$	$I\ (\langle a, b, c \rangle) = \langle \sqrt{2b^2 + 2c^2 - a^2}, b, c \rangle$	$O\ (\ ) =$
	MR	If $\langle a', b', c' \rangle = \langle a, \sqrt{2a^2 + 2c^2 - b^2}, c \rangle$, then $' =$	$T = \mathcal{T}$	$I\ (\langle a, b, c \rangle) = \langle a, \sqrt{2a^2 + 2c^2 - b^2}, c \rangle$	$O\ (\ ) =$
	MR	If $\langle a', b', c' \rangle = \langle a, b, \sqrt{2a^2 + 2b^2 - c^2} \rangle$, then $' =$	$T = \mathcal{T}$	$I\ (\langle a, b, c \rangle) = \langle a, b, \sqrt{2a^2 + 2b^2 - c^2} \rangle$	$O\ (\ ) =$
	MR_1	If $\langle A'T$			
SMM					

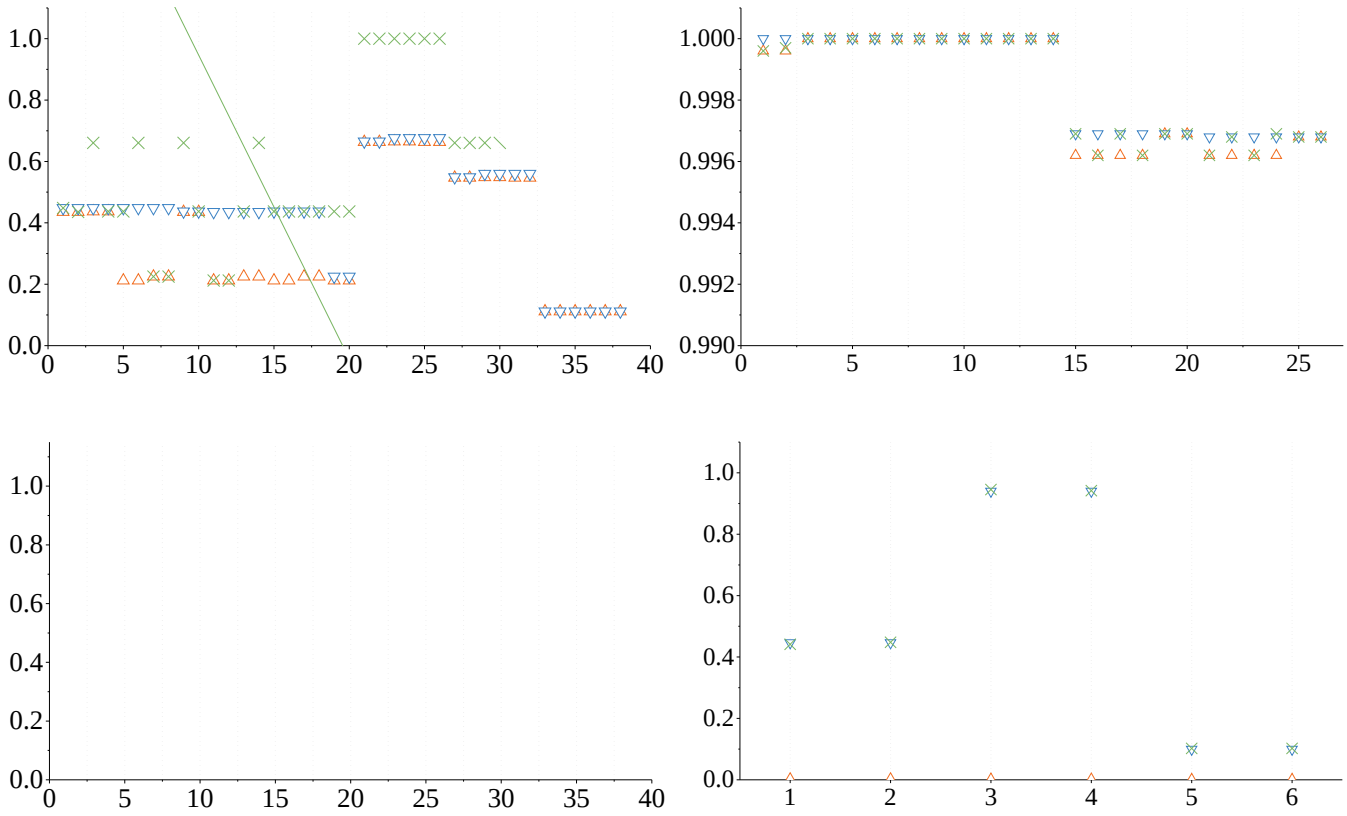

Fig. 1: Comparison among $min\{\theta \mid g(\theta)\}$, $max\{\theta \mid g(\theta)\}$, and θ.

step (a), the composition of MRs should not be complex. However, as a precaution to ensure that the generated composite MRs were valid with respect to their corresponding subject programs, we performed two tasks. Firstly, we conducted a desk check on these composite MRs, during which we detected no abnormality. Secondly, we tested all the four subject programs (their "original" versions; not their mutants) against these composite MRs. The testing results did not reveal any MR violation. To a large extent, these two tasks provide assurance that the composite MRs were correctly generated.

Subject program selection: Undoubtedly, it would be desirable to have a large set of programs for our empirical study. However, using a large set of programs was prohibited due to resource constraints. Nevertheless, using these four subject programs still provided a good insight into the validity of the hypothesis as stated at the beginning of Section 4.4 because these programs cover: (a) different application domains, including numerical calculations (*TSQ* and *SMM*), bioinformatics (*DNA*), and machine learning classifiers (*KNN*); and (b) different levels of complexity (*TSQ* and *KNN* are relatively less complex in logic, whereas *SMM* and *DNA* are relatively more complex).

Mutant generation and selection: The mutants used for *TSQ*, *SMM*, and *DNA* were sourced from other previous studies [8], [30], whereas the mutants used for *KNN* were generated by us (because we could not find mutants for this program from the published work). To a large extent, selecting extent,

TABLE 4: Mutants of Subject Programs

Sub. Pro.	Mu. ID	Code Change for Mutant Generation
TSQ	μ_1	Swap lines 6 and 8
	μ_2	Replace "p=(a+b+c)/2" by "p=(a+b+c)*2" in line 22
	μ_3	Replace "/2" by "*2" in lines 32, 40, and 48
	μ_4	Replace "(math.sqrt(3.0)*a*a)/4.0" by "(math.sqrt(3.0)*a*a)/2.0" in line 102
SMM	μ_1	Replace "n" by "1" in line 43
	μ_2	Replace "c[nz] = aij * b[k]" by "c[nz] = aij" in line 52
	μ_3	Replace "c[nz] = aij * b[k]" by "c[nz] = b[k]" in line 52
	μ_4	Replace "c[icol] += aij*b[k]" by "c[icol] += aij" in line 56
	μ	Replace "c[icol] += aij*b[k]" by "c[icol] += b[k]" in line 56
DNA	μ_1	Replace "ns=1<<G" by "ns=1<<C" in line 720 in file "seq.c"
	μ_2	Replace "ally[alias[i-1]-1]!=alias[i-1]" by "ally[alias[i-1]-1]>=alias[i-1]" in line 553 in file "seq.c"
	μ_3	Replace "i<b" by "i<=" in line 1097 in file "seq.c"
	μ_4	Replace "j=i+1" by "j=i-1" in line 555 in file "seq.c"
	μ	Replace "i<=(long)O" by "i>(long)O" in line 994 in file "seq.c"
	μ	Replace "itemp=alias[i-1]" by "itemp=alias[i+1]" in line 563 in file "seq.c"
	μ	Replace "j<=(long)O" by "j>=(long)O" in line 1119 in file "seq.c"
	μ	Replace "p->numsteps[i]+=weight[i]" by "p->numsteps[i]-=weight[i]" in line 946 in file "seq.c"
	$\mu_{\mathcal{A}_8}$	Replace "j<=(long)O" by "j>(long)O" in line 1126 in file "seq.c"
	μ_{10}	Replace "(i==j)" by "(i!=j)" in line 2700 in file "seq.c"
KNN	μ_1	Replace "-" by "/" in line 29 in function "euclideanDistance"
	μ_2	Replace "-" by "+" in line 29 in function "euclideanDistance"
	μ_3	Replace "+=" by "=" in line 29 in function "euclideanDistance"
	μ_4	Replace "math.sqrt(distance)" by "distance" in line 32 in function "euclideanDistance"
	μ	Replace "reverse=True" by "reverse=False" in line 41 in function "getNeighbors"
	μ	Swap lines 57 and 59 in function "getResponse"
	μ	Replace "+=" by "*=" in line 52 in function "getResponse"
	μ	Replace "+=" by "=" in line 52 in function "getResponse"
	$\mu_{\mathcal{A}_8}$	Replace "reverse=True " by "reverse=False" in line 55 in function "getResponse"

(a) both MR_1 and MR_2 belong to the special class of MRs in accordance with Definition 1;
(b) MR_1 is composable with MR_2 according to Definition 2;
(c) $I(T) = T$, I is bijective, and O is injective.

In the above guideline, condition (c) involves applying Theorems 2, 3, and 4 as discussed in Section 4. This condition also involves the hypothesis stated at the beginning of Section 4.4, which was confirmed by our empirical analysis (Section 5) to be highly likely to be held true. When the injectivity/bijectivity requirement of our guideline does not hold, testers should consider other information about the program under test, and the amount of testing resources available, to inform their own decisions on MR composition. For example, if the program has a long execution time, then the reduction in program executions achieved by using composite MRs (even at the expense of a slight deterioration in fault detection effectiveness) may still be a better choice.

Table 1 summaries the fault detection capability of MR_3 in four different testing scenarios. It can be seen from Table 1 that, in three of the four scenarios (Scenarios 1, 2, and 3), the fault detection capability of the composite MR (i.e., MR_3) is identical to those of applying both MR_1 and MR_2, if the three preconditions of the general guideline are satisfied. Since fewer test cases are required for testing MR_3 when compared with using both MR_1 and MR_2, the cost-effectiveness of MT in using MR_3 (instead of MR_1 and MR_2) is obviously improved in these three scenarios. Even in Scenario 4 where a definite conclusion on the fault detection capability of MR_3 (when compared with MR_1 and MR_2) cannot be drawn, we argue that it is very likely that the cost-effectiveness of MR_3 is higher than MR_1 and MR_2 because of two reasons: (a) the situation where $\theta_3 = 0$ should rarely occur according to our theoretical analysis (see Section 4.4), which was further confirmed to be true by our empirical analysis (see observation 1 in Section 5.5); and (b) our empirical analysis showed that $\theta_3 \geq min\{\theta_1, \theta_2\}$ for all cases and, in about three-quarter of the cases, we have $\theta_3 \geq max\{\theta_1, \theta_2\}$ (see observations 2 and 3 in Section 5.5).

6.2 Applicability of our General Guideline

To evaluate the practicality and usefulness of our general guideline on MR composition, we reviewed a set of published papers on MT through which the following two questions could be answered:

Q1: How likely is it that a given MR belongs to the special class, in accordance with Definition 1?

Q2: Given an MR that belongs to the special class, according to Definition 1, how likely is this MR to have a bijective input mapping I and an injective output mapping O?

We first studied in detail two recent survey papers on MT [37], [38], and then identified some other works on MT that have been developed after publishing those two surveys. After these exercises, we identified 10 popular application domains for MT as shown in Table 6. For each of these domains, we found some relevant published works related to MT. After a close examination, we compiled a list of MRs which were mentioned in these published works. Further checking of the list allowed us to identify some "common" MRs with similar types or characteristics within the same domain and even across different domains. These "common" MRs were counted only once in our review. For example, among the three MT-related papers on compilers, after tallying the count for "common" MRs, we found eight distinct MRs. We caution readers that, although our review did not (which was also infeasible to) involve every MT-related paper, we argue that the compiled list of MRs from our collected papers was fairly comprehensive because the list of MRs covered 10 different and popular domains for MT.

For Q1, we found 54.88% of MRs belong to the special class, in accordance with Definition 1. For Q2, we found that, among those MRs complying with Definition 1, 91.11% of them have a bijective I and an injective O. Based on these findings for Q1 and Q2, we can conclude that the three preconditions in our general MR composition guideline can easily be met ($50.00\% \quad 54.88\% \quad 91.11\%$). This shows that our general guideline should be widely applicable to many testing scenarios and application domains.

We also noted that the applicability of our guideline varies across different application domains. More specifically, according to the results in 6, the guideline is largely applicable to compilers, numeric and scientific programs, and AI systems (e.g., image processing and autonomous car systems), and relatively less applicable to biomedical applications, web services, embedded systems, and online

TABLE 5: Estimated Fault Detection Rates of Component and Composite MRs (where $\theta > 0$ and $\theta > 0$)

Sub. Pro.	Test Reference No.	Mu. ID	θ_x	θ_y	θ_{xy}
TSQ	1	μ_1	θ_1: 0.4482	θ_3: 0.4356	θ_{13}: 0.4482
	2	μ_1	θ_3: 0.4356	θ_1: 0.4482	θ_{31}: 0.4356
	3	μ_1	θ_1: 0.4482	θ : 0.4372	θ_1 : 0.6605
	4	μ_1	θ : 0.4372	θ_1: 0.4482	θ_1: 0.4372
	5	μ_1	θ_1: 0.4482	θ : 0.2123	θ_1 : 0.4372
	6	μ_1	θ : 0.2123	θ_1: 0.4482	θ_1: 0.6605
	7	μ_1	θ_1: 0.4482	θ : 0.2249	θ_1 : 0.2249
	8	μ_1	θ : 0.2249	θ_1: 0.4482	θ_1: 0.2249
	9	μ_1	θ_3: 0.4356	θ : 0.4372	θ_3 : 0.6605
	10	μ_1	θ : 0.4372	θ_3: 0.4356	θ_3: 0.4372
	11	μ_1	θ_3: 0.4356	θ : 0.2123	θ_3 : 0.2123
	12	μ_1	θ : 0.2123	θ_3: 0.4356	θ_3: 0.2123
	13	μ_1	θ_3: 0.4356	θ : 0.2249	θ_3 : 0.4372
	14	μ_1	θ : 0.2249	θ_3: 0.4356	θ_3: 0.6605
	15	μ_1	θ : 0.4372	θ : 0.2123	θ : 0.4372
	16	μ_1	θ : 0.2123	θ : 0.4372	θ : 0.4372
	17	μ_1	θ : 0.4372	θ : 0.2249	θ : 0.4372
	18	μ_1	θ : 0.2249	θ : 0.4372	θ : 0.4372
	19	μ_1	θ : 0.2123	θ : 0.2249	θ : 0.4372
	20	μ_1	θ : 0.2249	θ : 0.2123	θ : 0.4372
	21	μ_2	θ : 0.6652	θ : 0.6636	θ : 1.0
	22	μ_2	θ : 0.6636	θ : 0.6652	θ : 1.0
	23	μ_2	θ : 0.6652	θ : 0.6762	θ : 1.0
	24	μ_2	θ : 0.6762	θ : 0.6652	θ : 1.0
	25	μ_2	θ : 0.6636	θ : 0.6762	θ : 1.0
	26	μ_2	θ : 0.6762	θ : 0.6636	θ : 1.0
	27	μ_3	θ : 0.5487	θ : 0.5471	θ : 0.6605
	28	μ_3	θ : 0.5471	θ : 0.5487	θ : 0.6605
	29	μ_3	θ : 0.5487	θ : 0.5597	θ : 0.6605
	30	μ_3	θ : 0.5597	θ : 0.5487	θ : 0.6605
	31	μ_3	θ : 0.5471	θ : 0.5597	θ : 0.6605
	32	μ_3	θ : 0.5597	θ : 0.5471	θ : 0.6605
	33	μ_4	θ : 0.1115	θ : 0.1115	θ : 0.1115
	34	μ_4	θ : 0.1115	θ : 0.1115	θ : 0.1115
	35	μ_4	θ : 0.1115	θ : 0.1115	θ : 0.1115
	36	μ_4	θ : 0.1115	θ : 0.1115	θ : 0.1115
	37	μ_4	θ : 0.1115	θ : 0.1115	θ : 0.1115
	38	μ_4	θ : 0.1115	θ : 0.1115	θ : 0.1115
SMM	1	μ_1	θ_1: 0.9996	θ_2: 1.0	θ_{12}: 0.9996
	2	μ_1	θ_2: 1.0	θ_1: 0.9996	θ_{21}: 0.9997
	3	μ_2	θ_1: 1.0	θ_4: 1.0	θ_{14}: 1.0
	4	μ_2	θ_4: 1.0	θ_1: 1.0	θ_{41}: 1.0
	5	μ_2	θ_1: 1.0	θ : 1.0	θ_1 : 1.0
	6	μ_2	θ : 1.0	θ_1: 1.0	θ_1: 1.0
	7	μ_2	θ_4: 1.0	θ : 1.0	θ_4 : 1.0
	8	μ_2	θ : 1.0	θ_4: 1.0	θ_4: 1.0
	9	μ_3	θ_1: 1.0	θ : 1.0	θ_1 : 1.0
	10	μ_3	θ : 1.0	θ_1: 1.0	θ_1: 1.0
	11	μ_3	θ_1: 1.0	θ : 1.0	θ_1 : 1.0
	12	μ_3	θ : 1.0	θ_1: 1.0	θ_1: 1.0
	13	μ_3	θ : 1.0	θ : 1.0	θ : 1.0
	14	μ_3	θ : 1.0	θ : 1.0	θ : 1.0
	15	μ_4	θ_1: 0.9962	θ : 0.9969	θ_1 : 0.9969
	16	μ_4	θ : 0.9969	θ_1: 0.9962	θ_1: 0.9962
SMM	17	μ_4	θ_1: 0.9962	θ : 0.9969	θ_1 : 0.9969
	18	μ_4	θ : 0.9969	θ_1: 0.9962	θ_1: 0.9962
	19	μ_4	θ : 0.9969	θ : 0.9969	θ : 0.9969
	20	μ_4	θ : 0.9969	θ : 0.9969	θ : 0.9969
	21	μ	θ_1: 0.9962	θ_4: 0.9968	θ_{14}: 0.9962
	22	μ	θ_4: 0.9968	θ_1: 0.9962	θ_{41}: 0.9968
	23	μ	θ_1: 0.9962	θ : 0.9968	θ_1 : 0.9962
	24	μ	θ : 0.9968	θ_1: 0.9962	θ_1: 0.9969
	25	μ	θ_4: 0.9968	θ : 0.9968	θ_4 : 0.9968
	26	μ	θ : 0.9968	θ_4: 0.9968	θ_4: 0.9968
DNA	1	μ_2	θ_1: 0.0774	θ_2: 0.2085	θ_{12}: 0.2071
	2	μ_2	θ_2: 0.2085	θ_1: 0.0774	θ_{21}: 0.205
	3	μ_2	θ_3: 0.1116	θ_1: 0.0774	θ_{31}: 0.1444
	4	μ_2	θ_1: 0.0774	θ_4: 0.9936	θ_{14}: 0.9943
	5	μ_2	θ_4: 0.9936	θ_1: 0.0774	θ_{41}: 0.9938
	6	μ_2	θ_1: 0.0774	θ : 0.2522	θ_1 : 0.2543
	7	μ_2	θ : 0.2522	θ_1: 0.0774	θ_1: 0.2515
	8	μ_2	θ_3: 0.1116	θ_2: 0.2085	θ_{32}: 0.1116
	9	μ_2	θ_2: 0.2085	θ_4: 0.9936	θ_{24}: 0.9869
	10	μ_2	θ_4: 0.9936	θ_2: 0.2085	θ_{42}: 0.9799
	11	μ_2	θ_2: 0.2085	θ : 0.2522	θ_2 : 0.2434
	12	μ_2	θ : 0.2522	θ_2: 0.2085	θ_2: 0.8355
	13	μ_2	θ_3: 0.1116	θ_4: 0.9936	θ_{34}: 0.9948
	14	μ_2	θ_3: 0.1116	θ : 0.2522	θ_3 : 0.2567
	15	μ_2	θ_4: 0.9936	θ : 0.2522	θ_4 : 0.995
	16	μ_2	θ : 0.2522	θ_4: 0.9936	θ_4: 0.995
	17	μ_3	θ_3: 0.07	θ_2: 0.8059	θ_{32}: 0.07
	18	μ_4	θ_3: 0.0698	θ_2: 0.237	θ_{32}: 0.0698
	19	μ_4	θ_2: 0.237	θ_4: 1.0	θ_{24}: 1.0
	20	μ_4	θ_4: 1.0	θ_2: 0.237	θ_{42}: 1.0
	21	μ_4	θ_2: 0.237	θ : 0.261	θ_2 : 0.2851
	22	μ_4	θ : 0.261	θ_2: 0.237	θ_2: 0.9999
	23	μ_4	θ_3: 0.0698	θ_4: 1.0	θ_{34}: 1.0
	24	μ_4	θ_3: 0.0698	θ : 0.261	θ_3 : 0.2559
	25	μ_4	θ_4: 1.0	θ : 0.261	θ_4 : 1.0
	26	μ_4	θ : 0.261	θ_4: 1.0	θ_4: 1.0
	27	μ	θ_3: 0.214	θ_2: 0.9984	θ_{32}: 0.214
	28	μ	θ_3: 0.068	θ_2: 0.2307	θ_{32}: 0.068
	29	μ	θ_2: 0.2307	θ_4: 0.9735	θ_{24}: 0.9753
	30	μ	θ_4: 0.9735	θ_2: 0.2307	θ_{42}: 0.9753
	31	μ	θ_2: 0.2307	θ : 0.2704	θ_2 : 0.2809
	32	μ	θ : 0.2704	θ_2: 0.2307	θ_2: 0.2891
	33	μ	θ_3: 0.068	θ_4: 0.9735	θ_{34}: 0.9748
	34	μ	θ_3: 0.068	θ : 0.2704	θ_3 : 0.267
	35	μ	θ_4: 0.9735	θ : 0.2704	θ_4 : 0.986
	36	μ	θ : 0.2704	θ_4: 0.9735	θ_4: 0.986
	37	$\mu_{\mathcal{A}}$	θ_3: 0.214	θ_2: 1.0	θ_{32}: 0.214
	38	μ_{10}	θ_3: 0.0753	θ_2: 0.6398	θ_{32}: 0.0753
KNN	1	μ_3	θ_1: 0.0032	θ_3: 0.4465	θ_{13}: 0.4413
	2	μ_3	θ_3: 0.4465	θ_1: 0.0032	θ_{31}: 0.4479
	3	μ	θ_1: 0.0022	θ_2: 0.9401	θ_{12}: 0.9454
	4	μ	θ_2: 0.9401	θ_1: 0.0022	θ_{21}: 0.9418
	5	$\mu_{\mathcal{A}}$	θ_1: 0.0013	θ_2: 0.0995	θ_{12}: 0.1024
	6	$\mu_{\mathcal{A}}$	θ_2: 0.0995	θ_1: 0.0013	θ_{21}: 0.1022

search engines. We caution readers that this observation is based on the MRs reported in the literature. To date, only a relatively small set of MRs for compilers have been studied, and they all involve equality relations between source and follow-up outputs. This makes the guideline applicable to most of these MRs. Due to the nature of numeric and scientific application domains, the input and output mappings of the reported MRs are often composable and bijective, and therefore the guideline is also mostly applicable to these domains. Furthermore, image processing and autonomous driving systems are currently the main AI systems using MT. Most of the MRs defined for these systems are based on spatial transformation mappings that are composable and bijective, resulting in the applicability of the guideline in these domains. On the other hand, the reported MRs for biomedical applications, web services, embedded systems and online search engines often involve subset or substring relations, which are less likely to have bijective input and output mappings. The guideline, therefore, is less applicable to such systems. However, we also wish to highlight that this observation may vary as more MRs emerge.

TABLE 6: Applicability of the Guideline

Application Domains	References	No. of Identified MRs	No. of MRs complying with Def. 1	No. of MRs whose I is bijective and O is injective	Answer to Q1 (%)	Answer to Q2 (%)
Biomedical applications	[6], [7], [8]	42	12	12	28.57	100.00
Web services	[9], [10]	9	3	3	33.33	100.00
Embedded systems	[11], [12]	3	0	0	0.00	0.00
Component-based software	[13]	3	2	2	66.67	100.00
Compilers	[14], [15], [16]	8	8	5	100.00	62.50
Machine learning classifiers	[17], [18], [19], [20], [21]	16	7	7	43.75	100.00
Online search engines	[22], [23], [24]	9	0	0	0.00	0.00
Assorted computer science algorithms	[33], [34], [35]	27	16	11	59.26	68.75
Numerical and scientific programs	[30], [36]	17	14	14	82.35	100.00
AI systems (e.g., image processing and autonomous car systems)	[25], [26], [27], [28]	30	28	28	93.33	100.00
Total		**164**	**90**	**82**	**54.88**	**91.11**

6.3 Related Work

Two major challenges for MT are: (a) identification of MRs; and (b) additional computations of the program executions for follow-up test cases. Although MR composition can address both challenges, only few papers primarily focused on the effectiveness of MR composition — we found only two papers [30], [31] in this area. Obviously, MR composition reduces the number of program executions and, hence, lowers the computation costs — this is indisputable. However, these two studies on MR composition [30], [31] do not have a consensus on whether or not the fault detection capability after composition will be jeopardized. Furthermore, both studies [30], [31] adopted a purely empirical approach. Therefore, their observations could not provide a full picture of this issue and are dependent on the subjects being investigated. Understandably, some of their observations may look contradictory, that is, they reach different conclusions on the fault detection effectiveness of the composite MRs. On the other hand, with our theoretical results, such illusive contradictions are clarified. In summary, their results [30], [31] motivated our study, which in turn provides a more comprehensive interpretation of their results.

MR composition is an obvious and straightforward method to generate new MRs that is easily implemented and automated. However, this method requires the existence of some MRs for generation of new ones. Recently, with the increasing recognition and acceptance of MT by the software testing community, a growing number of research studies on MR generation / identification has emerged. Examples of these studies include machine-learning-based techniques [39], [40], [41], search-based techniques [42], [43], data-mutation-based techniques [26], [44], [45], pattern-based techniques [46], [47], and the category-choice approach [48], [49]. Since the main focus of this paper is not on MR generation, comparing our work with the above studies is beyond the scope of this paper.

7 Summary and Conclusion

Two major advantages of MR composition are the facilitation of automatic MR generation, and the reduction in testing costs (by reducing the number of program executions for follow-up test cases). However, MR composition has a potential drawback: The fault detection capability of the composite MR may be lower than that of its component MRs, jeopardizing the overall effectiveness of MT. This issue motivated us to perform theoretical and empirical analyses, with a goal of identifying characteristics that the component MRs should possess so that the fault detection capability of the generated composite MR will likely not be less than that of its component MRs. In short, given a pair of metamorphic relations MR and MR that belong to the special class defined in Definition 1, where MR is composable with MR , they should be used to form a composite MR if both the following conditions are met: (a) the output mapping of MR is injective; and (b) the input mapping of MR is a bijective mapping from the source inputs of MR to the source inputs of MR . This result is produced based on Theorems 1 to 4, Propositions 1 and 2, and the empirical analysis discussed in the paper. This result provides a solid foundation for MT and has paved a path for future studies on MR composition.

Based on our theoretical and empirical analyses, a convenient yet effective general guideline on MR composition has been developed. We further performed studies (by using a sample of MRs extracted from previously published works on MT) to confirm the applicability of the guideline across a range of application domains.

Our study has focused on a special class of MRs, as stated in Definition 1. Although this class of MRs is common, it would be worthwhile to extend our study to examine MRs that are outside this class. A hierarchy of composable MRs and their composite MRs could then be built. It would be interesting to investigate what further information and insights could be exploited from this hierarchy, using the theoretical results reported in this paper. This future study would enhance the foundation of MT research. Another potentially fruitful direction is a large-scale empirical analysis of the relationships among θ , $min f\theta$, θ g, and $max f\theta$, θ g. The results of this empirical analysis would help practitioners better estimate their testing costs.

Acknowledgments

We are indebted to Dr. Dave Towey of the University of Nottingham Ningbo China for his valuable comments and

suggestions on improving this paper. This work was supported in part by the National Natural Science Foundation of China (Grant Nos. 61772055 and 61872169), the Technical Foundation Project of Ministry of Industry and Information Technology of China (Grant No. JSZL2016601B003), and the Equipment Preliminary R&D Project of China (Grant No. 41402020102).

REFERENCES

[1] P. Ammann and J. Offutt, *Introduction to Software Testing*. New York, NY: Cambridge University Press, 2018.

[2] M. Pezzè and M. Young, *Software Testing and Analysis: Process, Principles, and Techniques*. Noida, India: Wiley, 2007.

[3] 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.

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

[5] F. T. Chan, T. Y. Chen, S. C. Cheung, M. F. Lau, and S. M. Yiu, "Application of metamorphic testing in numerical analysis," in *Proceedings of the IASTED International Conference on Software Engineering (SE'98)*, 1998, pp. 191–197.

[6] T. Y. Chen, J. W. K. Ho, H. Liu, and X. Xie, "An innovative approach for testing bioinformatics programs using metamorphic testing," *BMC Bioinformatics*, vol. 10, no. 1, 2009, Article no. 24.

[7] L. L. Pullum and O. Ozmen, "Early results from metamorphic testing of epidemiological models," in *Proceedings of ASE/IEEE International Conference on BioMedical Computing (BioMedCom)*, 2012, pp. 62–67.

[8] M. S. Sadi, F.-C. Kuo, J. W. K. Ho, M. A. Charleston, and T. Y. Chen, "Verification of phylogenetic inference programs using metamorphic testing," *Journal of Bioinformatics and Computational Biology*, vol. 9, no. 6, pp. 729–747, 2011.

[9] W. K. Chan, S. C. Cheung, and K. R. P. Leung, "A metamorphic testing approach for online testing of service-oriented software applications," *International Journal of Web Services Research (IJWSR)*, vol. 4, no. 2, pp. 61–81, 2007.

[10] C.-A. Sun, G. Wang, B. Mu, H. Liu, Z. S. Wang, and T. Y. Chen, "Metamorphic testing for web services: Framework and a case study," in *Proceedings of 2011 IEEE International Conference on Web Services*, 2011, pp. 283–290.

[11] W. K. Chan, T. Y. Chen, H. Lu, T. H. Tse, and S. S. Yau, "Integration testing of context-sensitive middleware-based applications: A metamorphic approach," *International Journal of Software Engineering and Knowledge Engineering*, vol. 16, no. 5, pp. 677–703, 2006.

[12] F.-C. Kuo, T. Y. Chen, and W. K. Tam, "Testing embedded software by metamorphic testing: A wireless metering system case study," in *Proceedings of 36th Conference on Local Computer Networks*, 2011, pp. 291–294.

[13] X.-L. Lu, Y.-W. Dong, and C. Luo, "Testing of component-based software: A metamorphic testing methodology," in *Proceedings of 7th International Conference on Ubiquitous Intelligence & Computing and 7th International Conference on Autonomic & Trusted Computing*, 2010, pp. 272–276.

[14] V. Le, M. Afshari, and Z. Su, "Compiler validation via equivalence modulo inputs," *ACM SIGPLAN Notices*, vol. 49, no. 6, pp. 216–226, 2014.

[15] Q. Tao, W. Wu, C. Zhao, and W. Shen, "An automatic testing approach for compiler based on metamorphic testing technique," in *Proceedings of 2010 Asia-Pacific Software Engineering Conference*, 2010, pp. 270–279.

[16] A. F. Donaldson, H. Evrard, A. Lascu, and P. Thomson, "Automated testing of graphics shader compilers," in *Proceedings of ACM on Programming Languages*, 2017, Article no. 93.

[17] C. Murphy, G. E. Kaiser, and L. Hu, "Properties of machine learning applications for use in metamorphic testing," Columbia University, U.S., Technical Report CUCS-011-08, 2011.

[18] X. Xie, J. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, "Application of metamorphic testing to supervised classifiers," in *Proceedings of 9th International Conference on Quality Software*, 2009, pp. 135–144.

[19] X. Xie, J. W. K. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, "Testing and validating machine learning classifiers by metamorphic testing," *Journal of Systems and Software*, vol. 84, no. 4, pp. 544–558, 2011.

[20] S. Nakajima and H. N. Bui, "Dataset coverage for testing machine learning computer programs," in *Proceedings of 23rd Asia-Pacific Software Engineering Conference*, 2016, pp. 297–304.

[21] P. Saha and U. Kanewala, "Fault detection effectiveness of metamorphic relations developed for testing supervised classifiers," in *Proceedings of IEEE International Conference on Artificial Intelligence Testing (AITest)*, 2019, pp. 157–164.

[22] Z. Q. Zhou, S. Zhang, M. Hagenbuchner, T. H. Tse, F.-C. Kuo, and T. Y. Chen, "Automated functional testing of online search services," *Software Testing, Verification and Reliability*, vol. 22, no. 4, pp. 221–243, 2012.

[23] Z. Q. Zhou, S. Xiang, and T. Y. Chen, "Metamorphic testing for software quality assessment: A study of search engines," *IEEE Transactions on Software Engineering*, vol. 42, no. 3, pp. 264–284, 2015.

[24] J. Brown, Z. Q. Zhou, and Y.-W. Chow, "Metamorphic testing of navigation software: A pilot study with Google maps," in *Proceedings of 51st Hawaii International Conference on System Sciences*, 2018, pp. 5687–5696.

[25] J. Mayer and R. Guderlei, "On random testing of image processing applications," in *Proceedings of 6th International Conference on Quality Software*, 2006, pp. 85–92.

[26] H. Zhu, D. Liu, I. Bayley, R. Harrison, and F. Cuzzolin, "Datamorphic testing: A method for testing intelligent applications," in *Proceedings of IEEE International Conference on Artificial Intelligence Testing (AITest)*, 2019, pp. 149–156.

[27] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems," in *Proceedings of 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 132–142.

[28] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of 40th International Conference on Software Engineering*, 2018, pp. 303–314.

[29] Z. Q. Zhou and L. Sun, "Metamorphic testing of driverless cars," *Communications of the ACM*, vol. 62, no. 3, pp. 61–67, 2019.

[30] G. Dong, B. Xu, L. Chen, C. Nie, and L. Wang, "Case studies on testing with compositional metamorphic relations," *Journal of Southeast University (English Edition)*, vol. 24, no. 4, pp. 437–443, 2008.

[31] H. Liu, X. Liu, and T. Y. Chen, "A new method for constructing metamorphic relations," in *Proceedings of 12th International Conference on Quality Software*, 2012, pp. 59–68.

[32] B. H. Smith and L. Williams, "On guiding the augmentation of an automated test suite via mutation analysis," *Empirical Software Engineering*, vol. 14, no. 3, pp. 341–369, 2009.

[33] M. Jiang, T. Y. Chen, F.-C. Kuo, and Z. Ding, "Testing central processing unit scheduling algorithms using metamorphic testing," in *Proceedings of 4th International Conference on Software Engineering and Service Science*, 2013, pp. 530–536.

[34] P. Rao, Z. Zheng, T. Y. Chen, N. Wang, and K. Cai, "Impacts of test suite's class imbalance on spectrum-based fault localization techniques," in *Proceedings of 13th International Conference on Quality Software*, 2013, pp. 260–267.

[35] 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.

[36] T. Y. Chen, J. Feng, and T. H. Tse, "Metamorphic testing of programs on partial differential equations: A case study," in *Proceedings of 26th Annual International Computer Software and Applications*, 2002, pp. 327–333.

[37] 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.

[38] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. H. Tse, and Z. Q. Zhou, "Metamorphic testing: A review of challenges and opportunities," *ACM Computing Surveys*, vol. 51, no. 1, article 4, pp. 1–27, 2018.

[39] U. Kanewala and J. M. Bieman, "Using machine learning techniques to detect metamorphic relations for programs without test oracles," in *Proceedings of 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, 2013, pp. 1–10.

[40] U. Kanewala, J. M. Bieman, and A. Ben-Hur, "Predicting metamorphic relations for testing scientific software: A machine learning approach using graph kernels," *Software Testing, Verification and Reliability*, vol. 26, no. 3, pp. 245–269, 2016.
[41] A. Nair, K. Meinke, and S. Eldh, "Leveraging mutants for automatic prediction of metamorphic relations using machine learning," in *Proceedings of 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation*, 2019, pp. 1–6.
[42] 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 29th ACM/IEEE International Conference on Automated Software Engineering*, 2014, pp. 701–712.
[43] B. Zhang, H. Zhang, J. Chen, D. Hao, and P. Moscato, "Automatic discovery and cleansing of numerical metamorphic relations," in *Proceedings of IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2019, pp. 235–245.
[44] H. Zhu, "JFuzz: A tool for automated Java unit testing based on data mutation and metamorphic testing methods," in *Proceedings of 2nd International Conference on Trustworthy Systems and Their Applications*, 2015, pp. 8–15.
[45] C.-A. Sun, Y. Liu, Z. Wang, and W. K. Chan, "μMT: A data mutation directed metamorphic relation acquisition methodology," in *Proceedings of 2016 IEEE/ACM 1st International Workshop on Metamorphic Testing (MET)*, 2016, pp. 12–18.
[46] 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, 2017.
[47] Z. Q. Zhou, L. Sun, T. Y. Chen, and D. Towey, "Metamorphic relations for enhancing system understanding and use," *IEEE Transactions on Software Engineering*, 2018. doi: 10.1109/TSE.2018.2876433.
[48] 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.
[49] C.-A. Sun, A. Fu, P.-L. Poon, X. Xie, H. Liu, and T. Y. Chen, "METRIC$^+$: A metamorphic relation identification technique based on input plus output domains," *IEEE Transactions on Software Engineering* (in press).

Kun Qiu received his B.S. in Automation from the Hefei University of Technology, Hefei, China, in 2013. He is currently a Ph.D candidate in the School of Automation Science and Electrical Engineering at Beihang University, Beijing, China. His research interests include software testing and software reliability analysis.

Zheng Zheng (SM' 18) received his Ph.D degree in computer software and theory from the Chinese Academy of Science, Beijing, China. In 2014, he worked as a research scholar in the Department of Electrical and Computer Engineering at Duke University, Durham, North Carolina, USA. He is currently a Professor at Beihang University, Beijing, China. His research interests include software dependability modeling, software testing, and software fault localization.

Tsong Yueh Chen (M' 03) received his Ph.D degree from The University of Melbourne. He is currently a Professor of Software Engineering at Swinburne University of Technology, Australia. Prior to joining Swinburne, he taught at The University of Hong Kong and The University of Melbourne. He is the inventor of metamorphic testing and adaptive random testing.

Pak-Lok Poon (M' 01) received his Ph.D degree in software engineering from The University of Melbourne. He is an Associate Professor with the School of Engineering and Technology, Central Queensland University, Australia. His research interests include software testing, requirements engineering and inspection, electronic commerce, and computers in education. He was a guest editor of the special issue of the *Journal of Systems and Software* on test oracles in 2018, and will be a guest editor of the special issue of the same journal on metamorphic testing in 2021. He was also an organizer for the *3rd, 4th, and 5th International Workshops on Metamorphic Testing* in 2018, 2019, and 2020, respectively.