μMT: A Data Mutation Directed Metamorphic Relation Acquisition Methodology

Chang-ai Sun*, Yiqiang Liu, and Zuoyi Wang
School of Computer and Communication Engineering
University of Science and Technology Beijing
Beijing 100083, China
casun@ustb.edu.cn; 1152138292@qq.com;
wzy100840@163.com

W.K. Chan
Department of Computer Science
City University of Hong Kong
Tat Chee Avenue, Hong Kong
wkchan@cityu.edu.hk

ABSTRACT
When figuring out the expected output for each test case is difficult, metamorphic testing can be applied to alleviate such situations. An involved key challenge is to derive metamorphic relations for the program under test. This paper proposes a data-mutation directed metamorphic relation acquisition methodology called μMT. Experimental results on three case studies show that μMT is feasible in deriving metamorphic relations for numeric applications and the derived metamorphic relations show reasonable fault detection effectiveness.

Categories and Subject Descriptors
D.2.4 [**Software Engineering**]: Software/Program Verification-*Reliability.*

General Terms
Algorithms, Reliability, Experimentation, Verification

Keywords
Metamorphic Testing, Test Oracle, Metamorphic Relation

1. INTRODUCTION
In software testing, testers execute program under test with a limited number of test cases and compare the actual outputs with the expected ones. When an inconsistency is detected, a fault is exposed, providing that the test cases are correct. Many existing testing techniques assume the availability of expected output (e.g., in the form of assertion statements in test cases) or failure conditions (e.g., crash) for each test case [6]. However, in some situations, it is difficult to decide whether the program outputs on test cases contain any failure, which is known as the test oracle problem [17]. For instance, given a sine function program *sin(x)*, it is easy to know the output of $sin(30^0)$ in mathematics is 0.5, but the value of $sin(31.5^0)$ in mathematics can only be approximated, say, by expecting a value failing within a small range of values. Determining the latter range remains an art of testing. The strength of such constraint also determines the ability of the test case to reveal program failures, which is a test oracle problem.

Metamorphic testing (MT) [4] has been proposed to alleviate the test oracle problem. The basic idea of MT is to leverage the relational properties of the program under test for test case generation and test outputs verification. Usually, these relational properties are *captured* by testers to be a set of metamorphic relations (MRs). For instance, one relational property of the sine function is $sin(x + 2*\pi) = sin(x)$, which indicates that given any two test cases x and x', if $x' = x + 2*\pi$, then the value of $sin(x)$ should be the same as the value of $sin(x')$. The input x is called the *source* test case, and x' is called the *follow-up* test case. If this assertion is violated by the actual outputs of a pair of test cases that satisfy the relation $x' = x + 2*\pi$, they reveal a failure.

To apply MT, a key challenge is to obtain MRs for the program under test. For instance, finding MRs for encryption programs is difficult [14]. Approaches (e.g., search-based MR inference [19], machine learning-based MR detection [8], and MR composition [9]) to address this challenge are emerging.

Independent to metamorphic testing, data mutation (DM) [12] has been proposed to address the test case generation problem. It is inspired by mutation testing to use some pre-defined DM operators to generate more test cases based on the seed test cases.

In this paper, we explore toward a solution for the MR acquisition problem by means of data mutation (DM). Specifically, we propose a semi-automated DM-directed MR acquisition methodology called μMT. The basic idea behind μMT consists of three parts. μMT uses specific DM operators to define the DM-*relations* among the input data of the test cases involved in an MR being constructed. It checks the validity of each DM-*relation* against the given constraints over the input domain of the program under test. μMT also uses the mapping rules provided by the tester of the program under test to define the expected relations on the actual outputs of the related test cases. We have developed an implementation of μMT, which has been integrated as a new component of MT4WS [15]. Finally, three case studies were conducted to evaluate the feasibility and fault detection effectiveness of the μMT methodology. The experimental results show that μMT provides a focal mechanism to direct testers to derive simple MRs, and the directed MRs demonstrates good and diverse fault detection effectiveness.

The main contribution made in this work is two-fold: (i) It proposes μMT, a novel methodology to guide the metamorphic relation acquisition through a combination of input data mutation with input domain constraint validation and output mapping rules. (ii) Case studies are reported to evaluate μMT on the fault detection effectiveness of the acquired MRs.

The rest of this paper is organized as follows. Section 2 introduces MT and DM. Section 3 presents our μMT methodology, and its supporting tool is described in Section 4. Section 5 presents the evaluation on μMT. Section 6 reviews related work, and the paper is concluded in Section 7.

* Corresponding author.

MET'16 co-located with ICSE 2016, May 14 - 22, 2016, Austin, USA.

http://dx.doi.org/10.1145/2896971.2896974

2. BACKGROUND

2.1 Metamorphic Testing

Metamorphic testing (MT) [4] uses the relational properties of the program under test to generate test cases and verify test outputs. These relational properties are captured as metamorphic relations.

A metamorphic relation (MR) may involve multiple inputs and even all but one output for these inputs. We restrict the scope of this paper to consider those MRs which each can be represented as (R, R_f), where R denotes the relation between source test case and follow-up test case, and R_f denotes the relation that the corresponding outputs should satisfy. We refer R to as the ***input relation*** and R_f to as the ***output relation***.

Figure 1 describes the principle of MT. Given a program under test P, a tester first derives a set of MRs based on the relational properties of P. Then, the tester reuses the previous test cases as a test suite T or generates T by employing any existing test case generation method. The tester derives another test suite T' via the relation R. Test cases in T and T' are referred to as ***source test cases*** and ***follow-up test cases***, respectively. A source test case and its follow-up test case form a ***metamorphic test group***. Next, the tester executes P using test cases from T and T' and get their outputs denoted as O and O', respectively. Lastly, the tester checks whether O and O' satisfy the relation R_f. If the relation R_f is violated, a failure is said to be detected.

MR is a core component in the MT methodology because both follow-up test case derivation and test outputs verification are based on MRs. Our previous work [13][14] has shown that the fault detection effectiveness of MT is heavily dependent on the MRs being used in the MT process.

Before applying MT to test a program, testers have to firstly derive MRs from the program under test. But, MR acquisition can be non-trivial, and thus become a key challenge in applying MT.

2.2 Data Mutation

Data mutation (DM) [11] is a test case generation technique inspired by mutation testing. Mutation testing [7] is a fault-based software testing technique, which has been widely used to evaluate the adequacy of test suites generated using a software test technique and compare the fault detection effectiveness of various testing techniques. Unlike mutation operators for mutation testing, which each operator injects an artificial fault into the program under test such that the created faulty version is *very similar* to the original version of the program, *data mutation operators* (DMOs) for data mutation transform the input data rather than the program under test. The original test cases are called ***seed test cases*** and the test cases derived by applying data mutation operators to seed test case are called ***mutant test cases***.

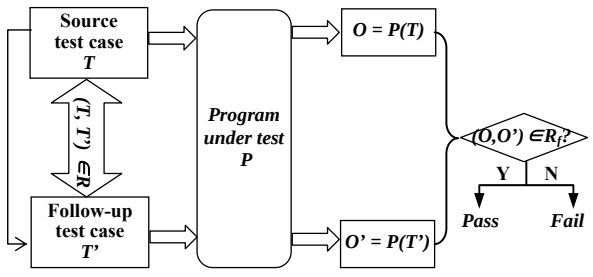

Figure 1. Principle of Metamorphic Testing over Two Test Cases

DM testing starts with preparing seed test cases (which can be generated using any test case generation method) and designing DMOs. The tester then applies these DMOs to the seed test cases to generate mutant test cases. Next, the tester executes the program under test with the seed test cases and their mutant test cases. The mutant test case is either dead or alive, depending on whether the output of the program over the corresponding seed test case is the same as that of the mutant. The tester then judges whether these DMOs are well designed by means of mutation scores. If the mutation score is not satisfactory, the tester can revise the seed test cases and the DMOs iteratively. Finally, the tester analyzes the correctness of program under test over these mutant test cases by looking into its observed behavior.

DM has been used to test an agent-oriented modeling environment CAMLE [12] and generate a large volume of messages needed by protocol testing [5]. DM has been observed to be an effective test case generation technique, especially for numerical programs.

3. DATA MUTATION DIRECTED ACQUISITIONS OF METAMORPHIC RELATIONS

3.1 The μMT Methodology

In this paper, we report an exploratory study on whether the use of DM has a potential to alleviate the MR acquisition problem. We propose μMT, a DM-directed MR acquisition methodology, and develop a tool to facilitate the application of μMT.

μMT is motivated by the following two observations. First, both MT and DM involve generating more test cases based on an original set of test cases: In MT, the input relation R in MRs is used for generating the *follow-up test cases* from the *source test cases*; and in DM, DMOs are used to generate the *mutant test cases* from the *seed test cases*. Second, both the input relation in an MR and DMOs are certain kinds of transformational operation (denoted as g) on the inputs and can be expressed as $x' = g(x)$.

An MR consists of two relations R and R_f. μMT leverages DMOs to guide the acquisition of the input relation R. Observing that DMOs, such as *double the value*, *increase by 1*, *decrease by 1*, or *swap two values*, are generic and "syntactic" transformational operation on inputs, μMT explores the possibility of using a set of mapping rules to construct the output relation R_f. In this paper, we propose to study whether it is feasible to formulate these mapping rules by identifying the property of the program under test.

Figure 2 shows an overview of μMT. μMT starts with selecting a set of data mutation operators (DMOs) (Step 1). Among these DMOs, the tester selects some of them for metamorphic testing through an analysis of the documentations of program under test P (Step 2). For instance, the tester can select the *swap* mutation operator for the triangle area calculation program. For the selected DMOs, the tester uses the source test cases as the seed test cases of DM, and applies these DMOs to generate mutant test cases (Step 3). Only those mutated test cases that are valid are selected into follow-up test cases. The validation on a mutant test case is verified by means of checking whether the test cases (input data) satisfy the given constraints over the input domain of the program under test (Step 4). For example, assuming that a source test case <0.5, 1, 1.2> is given, which denotes a triangle with three side lengths of 0.5, 1, and 1.2, and the operator "decrease by 1" is selected, the mutant test cases <-0.5, 1, 1.2>, <0.5, 0, 1.2>, and <0.5, 1, 0.2> are generated provided that the DMO changes only one parameter once. If this mutant test case violates some given constraint representing a valid triangle, then the mutant test case is

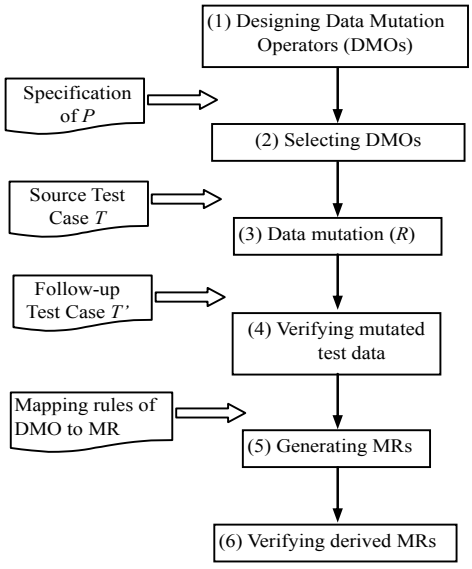

Figure 2. Overview of µMT

regarded as an invalid follow-up test case. (Note that the validity of the follow-up test case depends on the given set of constraints, and in theory, there is no need to require all the generated mutant test cases via a DMO should be all valid or all invalid.) Once the input relation of an MR is fixed, µMT generates the output relation R_f through the mapping rules (e.g., Table 1). Finally, the tester manual inspects the generated MRs to ensure that each generated MR is specific to some necessary property of the program under test (Step 6).

µMT is different from either the original MT or the original DM in the following aspects: (i) The acquisition of the input relation R in µMT is guided by data mutation operators; (ii) The acquisition of the output relation R_f is aided by the mapping rules from DMO to MR; (iii) The mapping rules are formulated based on some relational property of the program under test, which was not considered in the original DM. In the original DM, mutant test cases are said to be dead if the output of the seed is not the same as that of their mutants, but such identity relations are not always relational (and necessary) property of the program under test.

3.2 Mapping Rules for Output Relations

In this paper, we explore mapping rules specific to some necessary property of the program under test so that they may be helpful for a tester to acquire an MR efficiently. Suppose that the function of a program P is expressed as $f(x_1, \ldots, x_n)$, where $x_1, \ldots, x_n$ are the input parameters and $n > 0$. Consider the DMO "*swap*" operator which exchanges the value of x_i and x_j where $0 < i \neq j \leq n$, which can then be represented as follows:

$$swap(x_i, x_j) : x_i, x_j \rightarrow x_j, x_i \quad (1)$$

We then decide the output relation R_f based on the properties for the function f. If exchanging the order of input parameters should not affect the outputs of the function f, we then have the following mapping rule:

$$f(x_1,...,swap(x_i, x_j),...,x_n) = f(x_1,...,x_n) \quad (2)$$

Otherwise, we have the following mapping rule:

$$f(x_1,\ldots,swap(x_i, x_j),\ldots,x_n) \neq f(x_1 \ldots, x_n) \quad (3)$$

Consider the function for a rectangle area calculation program $f(a,b)$ where a and b denote length and width of the rectangle, respectively. Table 1 summarizes the mapping rules for the function $f(a,b)$ identified, where the DMO operators *Swap, Inc, Dec, Double, Halve* refer to swapping the two parameters, increase the parameter by 1, decrease the parameter by 1, double the value, and halve the value, respectively.

Table 1. Mapping rules of DMO to MR for the rectangle area calculation program (*a* and *b* are length and width)

No.	DMO	R_f in MR
1	*Swap*(*a*, *b*)	$f(Swap(a, b)) = f(a, b)$
2	*Inc* (*a*)	$f(Inc(a), b) > f(a,b)$
3	*Dec* (*a*)	$f(Dec(a),b) < f(a,b)$
4	*Double* (*a*)	$f(Double(a),b) = 2*f(a,b)$ or $f(Double(a),b) > f(a,b)$
5	*Halve* (*a*)	$f(Halve(a),b) = 1/2*f(a,b)$ or $f(Halve(a),b) < f(a,b)$

4. SUPPORTING TOOL

4.1 Existing Tool: MT4WS

In [14], we designed a language called MRDL to write MRs, which can express the way to generate follow-up test cases and the way to verify test outputs. We also developed a tool called MT4WS to automate MT for testing Web services. Figure 3 summarizes the architecture of MT4WS. For a given web service, WSDL Parser parses its WSDL interface to identify the input domains. Test Case Generator generates random test cases as the source test cases of MT. It imports the definitions of MRs written in MRDL, and Test Case Generator generates follow-up test cases. Executor then uses a text proxy to accept test cases generated above, invoke the Web service under test using these test cases, and intercepts the returned outputs. Evaluator then performs test output verifications based on the defined MRs. The tool finally reports a summary on the Web service under test, the used test cases, the used MRs, and whether each test is failed. MT4WS is extensible to handle other kinds of applications.

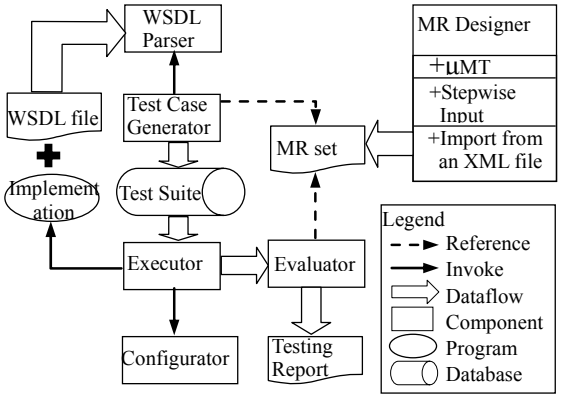

Figure 3. Architecture of enhanced MT4WS with µMT

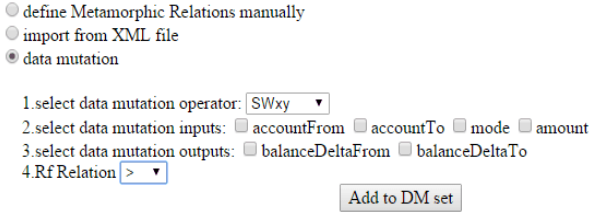

Figure 4. A snapshot of extended MT4WS supporting µMT

4.2 Our Component of MT4WS for μMT

To facilitate the application of μMT, we have enhanced MT4WS through extending its "MR Designer" component to support μMT. The extended parts of MT4WS are also shown in Figure 3.

Figure 4 shows a screen of MR Designer, which includes an option for using μMT (i.e. the "data mutation" option) to construct MRs. A tester firstly selects the "data mutation" option, the required data mutation operators (i.e. Step 1), the input parameters (i.e. Step 2), the output parameters (i.e. Step 3), and the relation of the output parameters (i.e. Step 4), and then presses the "Add to DM Set" button to generate an MR entry and append it into the generated MR set. The lists of options for Step 2 and Step 3 are dynamic texts that are extracted by the WSDL Parser based on the WSDL interface. Currently, we implemented the mapping rules presented in Table 1 in the component: Once a user selects a specific data mutation operator, the tool generates its corresponding MR using the template mapping rules, which are pre-determined manually and codified in the component beforehand. In future, we will support newly defined mapping rules by extending the "Stepwise Input" option in the "MR Designer" component. Interested readers may refer to the thesis [16] of an author (Wang) of this paper for more details on the design and implementation of this enhancement.

5. EMPIRICAL STUDY

In this section, we report three case studies to validate the feasibility of μMT and the effectiveness of the constructed MRs.

5.1 Research questions

We aim to answer the following three research questions:

- ***RQ1****: To what extent can μMT derive MRs?* We measured the number of MRs derived by μMT for this purpose.
- ***RQ2****: How effectively can the MRs derived by μMT detect faults?* We measured the fault detection effectiveness of these MRs in term of mutation score.
- ***RQ3***: *Is there any correlation between the MRs derived via different data mutation operators and their fault detection effectiveness?* We compared the mutation scores of the derived MRs among data mutation operators.

5.2 Experimental design and procedure

5.2.1 Subject programs and seeded faults

Three programs modeled after the main operations of real-life applications were selected as subject programs. We used mutation analysis [1] (precisely mutation score) to measure the fault detection effectiveness of the derived MRs using μMT. All these subject programs were written in Java by our research group, and each implemented a service port (operation) of a Web service.

ATM offered features, such as withdrawal, deposit, transfer, and query [13][14]. Among these features, the transfer operation was selected as the first subject program, which calculated the commission fee based on the China Agriculture Bank's policy.

BillCal calculated the customer's monthly bill according to China Unicom's billing policy [16]. The program accepted the input data including call time, free base call time, call rate, flow, free base flow, flow rate, and monthly base fee. Different monthly package types (A or B) had different amounts of free base call time, free base flow, call rate, and flow rate.

BaggBill was an aviation checked baggage billing service based on Air China's baggage billing standard [16]. For different air-ticket classes, passengers were allowed to carry certain limited pieces and weights of luggage, and overweight or additional luggage had to be charged based on the billing policy.

We employed MuJava [11] for generating mutants to inject seeded faults into the program code. We used the method-level mutation operators (i.e. AORB, AORS, AOIU, AOIS, AODU, AODS, ROR, COR, COD, COI, SOR, LOR, LOI, LOD, and ASRS.) to create program mutants as they are general. Each program mutant was created by inserting only a single fault into the program code of each subject. In total, we generated 163, 112, and 67 mutants for the three subject programs, respectively.

Table 2 provides a summary of subject programs and seeded faults. Columns "Description" and "Lines of Code" show the functionality and size of the subject programs, respectively, and Column "Number of Seeded Faults" and "Applicable Mutation Operators" show the number of generated mutants and method level mutation operators that are used for mutation. The latter is mainly limited to the operations in the subject program.

5.2.2 Fault detection effectiveness metrics

Mutation score (MS) is referred to as the ratio of the number of mutants that are killed against the total number of nonequivalent mutants. It can be formally defined as:

$$MS(P, TS) = N_k / (N_m - N_e) \quad (4)$$

where P is the program under test, TS is a test suite, N_k is the number of mutants killed by TS, N_m is the total number of mutants, and N_e is the number of equivalent mutants. An equivalent mutant refers to a mutant that outputs the same outputs for every test case of TS. MS indicates the fault detection effectiveness of a test suite. That is, the higher the mutation score is, the more effective the test suite will be. For each mutant that was not killed by any test case, we manually checked whether it is an equivalent mutant by multiple researchers to reduce their impact on what we measure on fault detection effectiveness. Finally, we excluded those confirmed equivalent mutants from our experiments.

5.2.3 Derivation of Metamorphic Relations

With the documentation of each subject program, we first determined which DMOs (the four operators described in Section 3.2) were applicable, then considered their relational properties, finally applied the mapping rule of each DMO to construct the output relation of the MR being constructed. In this way, DMO provides a focal methodology for testers to assess the applicability of each MR being constructed. The mapping rules were used as templates for MR definition. As a result, we have derived 4, 32, and 36 MRs for the three subject programs, respectively.

Table 2. Summary of subject programs and seeded faults

Program	Description	Line of Code	Number of Seeded Faults	Applicable Mutation Operators
ATM	transfer operation of China Agriculture Bank	263	163	AORB, AOIU, AOIS, ROR, COR, COD, COI, LOI
BillCal	monthly bill calculation of China Unicom	112	112	AORB,AOIU, AOIS, ROR, COR, COI, LOI
BaggBill	checked baggage billing of Air China	215	67	AORB, AOIU, AOIS, ROR, COI, LOI

5.2.4 Test case generation

Based the specification, we randomly generated a set of source test cases. For *ATM*, we chose the sizes of metamorphic test groups as 10, 20, 50, and 100 for each MR, and for both *BillCal* and *BaggBill*, we chose 1, 5, 10, and 20 for each MR. This was because that *ATM* was more complex that *BillCal* and *BaggBill*, thus having more mutants.

Also, all three subject programs had constraints on the input domains. For example, in the first case study, the amount in the transfer operation ranged from 0 to 50000, and the transfer mode was limited to 0..3; in the second case study, once one parameter (i.e. *planType*) was fixed, other parameters (such as flowBench, flowPer) had their corresponding scopes; similarly, in the third case study, when one parameter (i.e. *airClass*) was fixed, other parameters (such as *benchmark*) had their corresponding scopes.

5.2.5 Data collection

We executed the subject programs with the generated test cases. After the execution, we checked whether the MR was violated. In details, assuming that MR_i was used to derive the follow-up test cases, then we checked whether the outputs of source test cases and follow-up test cases satisfied the corresponding output relation of MR_i. A mutant was said to be "killed" provided that there was a metamorphic test group whose execution resulted in violation of the corresponding MR. We then calculated the mutation score of each test suite used.

Table 3: Transfer fee calculation

	I	II	III	IV
Charge Percentage	0%	0.5%	0.5%	1%
Min(￥)	0	1	1	1
Max(￥)	0	50	50	50
Limit Per Transfer (￥)	50000	50000	50000	50000

5.3 Results and Analysis

(1) Answering RQ1 (applicability): In our experiments, μMT was employed to derive a large number of MRs. We found that these MRs were often simple. μMT is demonstrated to be feasible.

We use the *ATM* program to illustrate the MR derivation process. The specification of the transfer operation is shown in Table 3. The input of the transfer operation is represented as a four-tuple $<A, B, P, M>$, where A is referred to as the transfer-out account; B is referred to as the transfer-in account; P is referred to as the transfer mode (for instance, I referring to the transfer between two accounts in the same bank and city); and M is referred to as the amount ranging from 0 to 50000. The transfer fee is charged from the transfer-out account. The output of the transfer operation is represented as a couple $<\triangle A, \triangle B>$, where $\triangle A$ refers to as the balance difference of the account A before the transfer and after the transfer, and $\triangle B$ refers to as the balance difference of the account B due to the transfer.

Applicable data mutation operators for the transfer include Inc (i.e. increase by 1), Dec (i.e. decrease by 1), Double, and Halve. Based on these data mutation operators, we define their mapping rules of DMO to MR. For instance, for the transfer amount increasing by 1 (i.e. applying the Inc operator), the balance difference for the account B should increase by a certain amount. By using the mapping rule of "*f(Inc(M), A, B, P) > f(M), A, B, P)*", we have MR1 as shown in Table 4. Similarly, we can derive the remaining MRs through applying the other three data mutation operators.

Table 5. Mutation scores on the *ATM* program

		MS (%)			
DMO	**MR**	**TC=10**	**TC=20**	**TC=50**	**TC=100**
Inc	MR1	6.75	7.98	7.98	8.59
Dec	MR2	6.75	7.98	7.98	9.20
Double	MR3	14.11	15.34	16.56	17.79
Halve	MR4	14.11	15.95	16.56	16.56

Table 6. Mutation scores on the *BillCal* program

		MS (%)			
DMO	**MR**	**TC=1**	**TC=5**	**TC=10**	**TC=20**
Inc	MR1	15.18	15.18	15.18	15.18
	MR5	14.29	15.18	15.18	16.07
	MR9	15.18	15.18	15.18	15.18
	MR13	14.29	15.18	15.18	15.18
	MR17	10.71	13.39	14.29	14.29
	MR21	9.82	11.61	13.39	14.29
	MR25	8.93	9.82	9.82	10.71
	MR29	8.93	9.82	9.82	9.82
Dec	MR2	15.18	15.18	15.18	15.18
	MR6	14.29	15.18	15.18	15.18
	MR10	15.18	15.18	15.18	15.18
	MR14	14.29	15.18	15.18	15.18
	MR18	9.82	10.71	11.61	14.29
	MR22	8.93	11.61	14.29	15.18
	MR26	8.93	9.82	10.71	11.61
	MR30	9.82	9.82	9.82	10.71
Double	MR3	5.36	6.25	6.25	6.25
	MR7	5.36	7.14	7.14	7.14
	MR11	5.36	7.14	7.14	7.14
	MR15	5.36	5.36	5.36	5.36
	MR19	5.36	6.25	6.25	6.25
	MR23	4.46	6.25	6.25	6.25
	MR27	4.46	6.25	6.25	6.25
	MR31	5.36	6.25	6.25	6.25
Halve	MR4	5.36	6.25	6.25	6.25
	MR8	7.14	7.14	7.14	7.14
	MR12	5.36	7.14	7.14	7.14
	MR16	4.46	4.46	5.36	5.36
	MR20	6.25	6.25	7.14	7.14
	MR24	5.36	7.14	7.14	7.14
	MR28	3.57	6.25	6.25	8.04
	MR32	4.46	7.14	7.14	7.14

Table 4. Derived MRs for ATM using μMT

MR	**DMO**	**Mapping Rule**	***MR***	
			R	R_f
MR1	*Inc*	$f(Inc(M), A, B, P) > f(M), A, B, P)$	M'=M+1	$\triangle B' > \triangle B$
MR2	*Dec*	$f(Dec(M), A, B, P) < f(M), A, B, P)$	M'=M-1	$\triangle B' < \triangle B$
MR3	*Double*	$f(Double(M), A, B, P) \geq k * f(M), A, B, P)$	M'=2*M	$\triangle B' \geq 2*\triangle B$
MR4	*Halve*	$f(Half(M), A, B, P) \leq k * f(M), A, B, P)$	M'=0.5*M	$\triangle B' \leq 0.5*\triangle B$

Table 7. Mutation scores on the *BaggBill* program

		MS (%)			
DMO	**MR**	**TC=1**	**TC=5**	**TC=10**	**TC=20**
Inc	MR1	4.48	5.97	7.46	7.46
	MR5	4.48	4.48	8.96	8.96
	MR9	4.48	5.97	7.46	7.46
	MR13	7.46	10.45	14.93	16.42
	MR17	4.48	4.48	4.48	7.46
	MR21	5.97	5.97	5.97	7.46
	MR25	4.48	10.45	11.94	13.43
	MR29	10.45	17.91	22.39	23.88
	MR33	4.48	7.46	7.46	13.43
Dec	MR2	4.48	4.48	5.97	7.46
	MR6	5.97	5.97	5.97	8.96
	MR10	5.97	5.97	7.46	7.46
	MR14	8.96	11.94	16.42	16.42
	MR18	4.48	5.97	7.46	7.46
	MR22	4.48	4.48	7.46	8.96
	MR26	4.48	7.46	8.96	11.94
	MR30	10.45	16.42	22.39	22.39
	MR34	4.48	5.97	7.46	7.46
Double	MR3	7.46	8.96	10.45	11.94
	MR7	5.97	8.96	8.96	10.45
	MR11	7.46	8.96	8.96	10.45
	MR15	11.94	17.91	17.91	19.40
	MR19	5.97	8.96	8.96	8.96
	MR23	7.46	7.46	10.45	10.45
	MR27	8.96	11.94	17.91	17.91
	MR31	16.42	23.88	28.36	28.36
	MR35	7.46	13.43	13.43	13.43
Halve	MR4	4.48	8.96	10.45	10.45
	MR8	8.96	10.45	10.45	11.94
	MR12	5.97	8.96	11.94	13.43
	MR16	10.45	16.42	19.40	19.40
	MR20	5.97	7.46	8.96	8.96
	MR24	4.48	11.94	10.45	10.45
	MR28	8.96	11.94	16.42	22.39
	MR32	14.93	22.39	26.87	26.87
	MR36	7.46	10.45	14.93	17.91

We applied μMT to *BillCal* and *BaggBill* in the same way. The detailed process and the derived MRs will not be presented here due to page limit. Interested readers please refer to Wang's thesis [16] for full set of MRs constructed for these two subjects. In this paper, we refer to these MRs by their indexes as reported in [16].

(2) Answering RQ2 (fault detection effectiveness of μMT): The experimental data for three subject program are presented in Tables 5, 6, and 7, respectively. Note that the "DMO" column indicates the data mutation operators used to derive the corresponding MR; the "MR" column indicates which MR is used for testing; the "TC" column indicates the number of metaphoric test groups used for the corresponding MR. From Tables 5, 6, and 7, we have the following observations:

Different MRs demonstrate varying mutation scores. On *ATM*, MS varies from 6.75% to 17.79%; for *BillCal*, MS varies from 3.57% to 16.07; for *BaggBill*, MS varies from 4.48% to 28.36%.

The size of a test suite has an impact on the fault detection effectiveness of μMT. In general, a larger test suite has a higher mutation score, and the fault detection effectiveness improvement

Table 8. Average Mutation scores on the *BillCal* and *BaggBill* programs by DMO operators

	MS (%)							
BillCal								
TC	1		5		10		20	
DMO	Mean	SD	Mean	SD	Mean	SD	Mean	SD
Inc	12.17	2.82	13.17	2.42	13.50	2.36	13.84	2.29
Dec	12.05	2.90	12.83	2.57	13.39	2.29	14.06	1.83
Double	5.13	0.41	6.36	0.57	6.36	0.57	6.36	0.57
Halve	5.25	1.11	6.47	0.92	6.70	0.67	6.92	0.79
BaggBill								
TC	1		5		10		20	
DMO	Mean	SD	Mean	SD	Mean	SD	Mean	SD
Inc	12.17	2.82	13.17	2.42	13.50	2.36	13.84	2.29
Dec	12.05	2.90	12.83	2.57	13.39	2.29	14.06	1.83
Double	5.13	0.41	6.36	0.57	6.36	0.57	6.36	0.57
Halve	5.25	1.11	6.47	0.92	6.70	0.67	6.92	0.79

is inconsistent with the increasing size of test suites for three case studies. On each subject program, generally speaking, increasing the size of a test suite shows a steady increase in MS. However, all subject programs show that when the MS can easily be saturated after running a small number of test cases.

Among these 72 MRs derived using μMT reported in this paper, when using 20 test cases as source test cases, their average mutation score ranges from 6.25% to 28.36%. In summary, the simple MR constructed via μMT demonstrates good and diverse fault detection effectiveness. This diverse effectiveness appears indicating that these MRs can detect different seeded faults.

(3) Answer to RQ3 (correlation on fault detection effectiveness between DMOs and MRs): DMOs that are used to derive MRs affect the fault detection effectiveness of μMT. On *ATM*, each DMO can only lead to the construction of one MR, but there are multiple MRs in the other two subject programs. For ease of our study on the impact of DMOs on mutation scores, we also summarize the average mutation scores of μMT at the DMO level for *BillCal* and *BaggBill* in Table 8. (We note that Table 5 already serves as the same summary for *ATM*.)

For the same size of test suites in all three case studies, *Inc* and *Dec* always have similar mutation scores, and *Double* and *Halve* have similar mutation scores. However, *Inc* and *Dec* have higher mutation scores than *Double* and *Halve* on *BillCal*, but this observation is reversed on *ATM* and *BaggBill*. Moreover, even for MRs derived using the same mutation operators, they may show significantly different mutation scores. For example, both MR1 and MR29 in Table 7 are derived using the *Inc* operator, their mutation scores are different even for the same size of test suites. Therefore, we concluded that the μMT methodology to use DMOs does not reveal that there is any consistent fault detection effectiveness in the DMO aspect.

6. RELATED WORK

MT [10] has detected faults in previously extensively tested, real-life programs [18]. A crucial component of MT is the MRs, since MRs are used to generate follow-up test cases and to verify test results. We introduce closely related work on the MR acquisition.

Kanewala and Bieman [8] presented a machine learning approach to automatically detecting likely MRs of program functions using features extracted from a function's control flow graph. Their method provides a way to detect MRs using machine learning

technique. Their method is based on flow control graph of program under test and requires a number of sampling data for training, its approach may not be simple and efficient.

Zhang et al. [19] proposed a search-based approach to automatic inference of polynomial MRs for a program under test. Their method uses a set of parameters to represent a particular class of polynomial MRs, and turns the problem of inferring MRs into a problem of searching for suitable values of the parameters. Their method is useful to derive MRs for scientific applications. It may be very time-consuming for large interactive applications, since it requires multiple executions of the program under test.

Liu et al. [9] proposed a MR composition method which constructs new MRs from already identified ones. It was observed that the newly constructed relations were likely to deliver higher MT cost-effectiveness than the original ones. Thus, combining their method and μMT may produce more cost-effectiveness MRs.

Some research efforts are reported on how to identify *effective* MRs. Asrafi et al. [2] have observed a correlation between the code coverage achieved by an MR and its fault detection effectiveness. Liu et al. [10] found that a small number of diverse MRs could have a similar fault-detection ability to a test oracle

As reviewed in Section 4, we [15] have designed an XML-based metamorphic relation description language. The supporting tool for μMT presented in this work has been integrated into MT4WS to further improve the efficiency and automation of MT.

We [3] have also studied the use of metamorphic service in support of the metamorphic testing of web services.

7. CONCLUSION

Identifying MRs is an important yet difficult task for MT. In this work, we have proposed a novel methodology μMT to alleviate this difficulty. μMT uses data mutation to construct input relation and the generic mapping rule associated with each mutation operator to construct output relations. We have also presented a tool to support μMT. In the evaluation, we have observed that μMT is feasible to provide a focal mechanism to direct testers to derive simple MRs. In future work, we would like to explore more effective and efficient μMT methodology. We will study how to generate MRs with high code coverage and diversity. We plan to conduct a more extensive empirical evaluation of the fault detection effectiveness of the composition of MRs derived using μMT with more programs and data mutation operators.

8. ACKNOWLEDGMENTS

This research is supported in part by NSFC (no. 61370061), the Beijing Natural Science Foundation of China (Grant No. 4162040), the Fundamental Research Funds for the Central Universities (no. FRF-SD-12-015A), the Beijing Municipal Training Program for Excellent Talents (no. 2012D009006000002), and the GRF of Hong Kong RGC (nos. 111313, 11201114, and 11200015).

9. REFERENCES

[1] Andrews, J. H., Briand, L. C., and Labiche, Y. 2005. Is mutation an appropriate tool for testing experiments? In: *Proceedings of the 27th International Conference on Software Engineering*, 402–411.

[2] Asrafi, M., Liu, H. and Kuo, F.-C. 2011. On testing effectiveness of metamorphic relations: A case study. In: *Proceedings of the 5th International Conference on Secure System Integration and Reliability Improvement* (SSIRI 2011), 147-456.

[3] Chan, W.K., Cheung, S. C., and Leung, K.P.H., 2007. A metamorphic testing approach for online testing of service-oriented software applications. Journal of Web Services Research, 4(2):60-80.

[4] Chen, T. Y., Cheung, S. C. and Yiu, S. M., 1998. Metamorphic Testing: A New Approach for Generating Next Test Cases. *Technical Report*. HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology.

[5] Gu, S., Li, W., and Zhao, X. 2011. Brute force vulnerability testing technology based on data mutation. In: *Proceedings of 2011 IEEE Vehicular Technology Conference (VTC Fall 2011)*, IEEE Computer Society, 1-6.

[6] Harman, M., McMinn, P., Shahbaz, M. and Yoo, S., 2013. A Comprehensive Survey of Trends in Oracles for Software Testing. *Technical Report*. CS-13-01, Department of Computer Science, University of Sheffield.

[7] Jia, Y. and Harman, M. 2010. An analysis and survey of the development of mutation testing. *IEEE Transactions. Software Engineering,* 37(5): 649-678.

[8] Kanewala, U. and Bieman, J. M. 2013. Using machine learning techniques to detect metamorphic relations for programs without test oracles. In: *Proceedings of the 24th IEEE International Symposium on Software Reliability Engineering* (ISSRE '13), 1–10.

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

[10] Liu, H., Kuo, F.-C., Towey, D., and Chen, T. Y. 2014. How effectively does metamorphic testing alleviate the oracle problem? *IEEE Transactions on Software Engineering*, 40(1):4-22.

[11] Ma,Y. S., Offutt, J. and Kwon, Y. R. 2006. MuJava: a mutation system for Java. In: *Proceedings of the 28th International Conference on Software Engineering (ICSE 2006)*, 827–830.

[12] Shan, L. and Zhu, H. 2009. Generating structurally complex test cases by data mutation: a case study of testing an automated modelling tool. *The Computer Journal.* 52(5):571-588.

[13] Sun, C., Wang, G., Mu, B.H., Liu, H., Wang, Z.S., and Chen, T.Y. 2012. A Metamorphic Relation-Based Approach to Testing Web Services without Oracles. *International Journal of Web Services Research*, 9(2):51-73.

[14] Sun, C., Wang, Z. and Wang, G. 2014. A property-based testing framework for encryption programs. *Frontiers of Computer Science,* 8(3): 478-489.

[15] Sun, C., Wang, G., Wen, Q., Towey, D., and Chen, T.Y. 2015. MT4WS: An Automated Metamorphic Testing System for Web Services. *International Journal of High Performance Computing and Networking*, In press

[16] Wang, Z. 2014. Research on Data Mutation Based Metamorphic Relation Acquisition Technique. *Master Thesis*. University of Science and Technology Beijing, Beijing, China.

[17] Weyuker, E. J. 1982. On Testing Non-Testable Programs. *The Computer Journal*, 25(4):465–470.

[18] Xie, X., Wong, E., Chen, T.Y., and Xu B. 2013. Metamorphic Slice: An Application in Spectrum-based Fault Localization. *Information and Software Technology*, 55(5):866-879.

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