Journal of Southeast University (English Edition) Vol. 24, No. 4, pp. 437 – 443 Dec. 2008 ISSN 1003—7985

Case studies on testing with compositional metamorphic relations

Dong Guowei Xu Baowen Chen Lin Nie Changhai Wang Lulu

(School of Computer Science and Engineering, Southeast University, Nanjing 210096, China)
(Jiangsu Institute of Software Quality, Nanjing 210096, China)

Abstract: Some metamorphic relations (MR) are not good at detecting faults in metamorphic testing. In this paper, the method of making compositional MR (CMR) based on the speculative law of proposition logic is presented. This method constructs new MRs by composing existing MRs in a pairwise way. Because CMR contains all the advantages of the MRs that form it, its fault detection performance is wonderful. On the other hand, the number of relations will decrease greatly after composing, so a program can be tested with much fewer test cases when CMRs are used. In order to research the characteristics of a CMR, two case studies are analyzed. The experimental results show that the CMR's performance is mostly determined by the central MRs forming it and the sequence of composition. Testing efficiency is improved greatly when CMRs are used.

Key words: software testing; metamorphic testing; compositional metamorphic relation

Software testing can be used for finding and correcting errors in software. But sometimes, there are oracle problems[1] in testing; that is, it is difficult to decide whether the results of the program under testing agrees with the expected results. To solve this problem, Chen presented metamorphic testing(MT)[2]. This method tests software by checking relations among execution results, and does not call for expected outputs.

There are two prominent traits for MT: 1) To check execution results, metamorphic relations (MR)[2] should be constructed; 2) To attain original test cases[2], MT should be used with other test case generating methods.

Chen et al. reported on the MT of programs for solving partial differential equations[3]; Gotlieb et al. developed an automated framework to check against a restricted class of MRs[4]; Zhou et al. introduced ways to design MRs for non-numerical problems[5]; Tse et al. applied a metamorphic approach to the unit testing[6] and integration testing[7] of context-sensitive middleware-based applications; MT has also been used in testing SOA software[8-9]; Chen et al. introduced how to select useful MRs by a case study[10]; Chen et al. investigated the integration of MT with global symbolic evaluation[11] and fault-based testing[12]; Wu et al. gave the evaluation and comparison of special case testing, MT with special and random test cases[13], and put forward an iterative MT technique[14]. These researches validate that MT is good for solving oracle problems, but most of them only consider a program's function, so blindness is hard to avoid.

In this paper, we provide the definition of CMR (compositional MR) based on the speculative law of proposition logic, and prove that these new relations are useful for fault detection by two case studies, SpMatMul and TriSquare. Some conclusions for CMR construction are also presented.

1 Concepts

Definition 1 (metamorphic relation) Suppose that program P computes function f, which is also referred to as the specification that P must accord with. Let $x_1, x_2, \ldots, x_n (n > 1)$ be n different inputs of function f, if their satisfaction of relation r implies that their corresponding outputs $f(x_1), f(x_2), \ldots, f(x_n)$ satisfy relation r_f, that is

$$r(x_1, x_2, \ldots, x_n) \Rightarrow r_f(f(x_1), f(x_2), \ldots, f(x_n)) \quad (1)$$

then (r, r_f) is a metamorphic relation of f. Because P is an implementation of f, if P is correct, it should also satisfy this relation, that is

$$r(I_1, I_2, \ldots, I_n) \Rightarrow r_f(P(I_1), P(I_2), \ldots, P(I_n)) \quad (2)$$

where $I_1, I_2, \ldots, I_n$ are inputs of P that associate with $x_1, x_2, \ldots, x_n$, and $P(I_1), P(I_2), \ldots, P(I_n)$ are their outputs. So (r, r_f) is also an MR of P. When we test P with (r, r_f), the test cases originally given are original test cases (OTC), others which are deduced from relation r are follow-up test cases (FTC).

Definition 2 (compositional MR) Let $(r_1, r_{f1}), (r_2, r_{f2}), \ldots, (r_q, r_{fq})$ be q MRs of program P, where

$$\begin{aligned} &r_1(I_1^1, I_2^1, \ldots, I_{n1}^1) \Rightarrow r_{f1}(P(I_1^1), P(I_2^1), \ldots, P(I_{n1}^1)) \\ &r_2(I_1^2, I_2^2, \ldots, I_{n2}^2) \Rightarrow r_{f2}(P(I_1^2), P(I_2^2), \ldots, P(I_{n2}^2)) \\ &\quad\vdots \\ &r_q(I_1^q, I_2^q, \ldots, I_{nq}^q) \Rightarrow r_{fq}(P(I_1^q), P(I_2^q), \ldots, P(I_{nq}^q)) \end{aligned}$$

We first select two relations, (r_i, r_{fi}) and (r_j, r_{fj}). According to the speculative law of proposition logic, the following formula can be deduced:

$$\begin{aligned} &[r_i(I_1^i, I_2^i, \ldots, I_{ni}^i) \wedge r_j(I_1^j, I_2^j, \ldots, I_{nj}^j)] \Rightarrow \\ &\quad [r_{fi}(P(I_1^i), P(I_2^i), \ldots, P(I_{ni}^i)) \wedge \\ &\quad r_f(P(I_1^j), P(I_2^j), \ldots, P(I_{nj}^j))] \end{aligned} \quad (3)$$

If expression (3) can be evolved to

$$r_{\text{temp}}(I_1, I_2, \ldots, I_{\text{temp}}) \Rightarrow r_{\text{ftemp}}(P(I_1), P(I_2), \ldots, P(I_{\text{temp}})) \quad (4)$$

by techniques such as variable replacement, equivalent transformation, and so on, then the next composition is done with $(r_{\text{temp}}, r_{\text{ftemp}})$ and another MR that has not been selected. Re-

Received 2008-03-20.

Biographies: Dong Guowei (1983—), male, graduate, dgw@seu.edu.cn; Xu Baowen (corresponding author), male, doctor, professor, bwxu@seu.edu.cn.

Foundation items: The National Natural Science Foundation of China (No. 60425206, 60633010, 60773104, 60503033), the Excellent Talent Foundation of Teaching and Research of Southeast University.

Citation: Dong Guowei, Xu Baowen, Chen Lin, et al. Case studies on testing with compositional metamorphic relations[J]. Journal of Southeast University (English Edition), 2008, 24(4): 437 – 443.

peat this pairwise composition $(q-1)$ times. If we can get an MR, (r_c, r_{fc}) finally, where

$$r_c(I_1, I_2, \ldots, I_c) \Rightarrow r_{fc}(P(I_1), P(I_2), \ldots, P(I_c)) \quad (5)$$

then (r_c, r_{fc}) is a compositional MR for all q MRs.

For example, Square(double, double) is a program for calculating rectangular squares:

```
double Square(double a, double b){
    return a * b;
}
```

We construct two MRs, (r_1, r_{f1}) and (r_2, r_{f2}) for it, where r_1 is $[(a', b') = (b, a)]$, r_{f1} is $[\text{Square}(a', b') = \text{Square}(a, b)]$, r_2 is $[(a', b') = (2a, 2b)]$, and r_{f2} is $[\text{Square}(a', b') = 4 \times \text{Square}(a, b)]$. It is obvious that

$$[(a_2, b_2) = (b_1, a_1)] \wedge [(a_1, b_1) = (2a, 2b)] \Rightarrow [\text{Square}(a_2, b_2) = \text{Square}(a_1, b_1)] \wedge [\text{Square}(a_1, b_1) = 4 \times \text{Square}(a, b)]$$

After deducing, we get a CMR, (r_c, r_{fc}) of mr_1 and mr_2, where r_c is $[(a_c, b_c) = (2b, 2a)]$, and r_{fc} is $[\text{Square}(a_c, b_c) = 4 \times \text{Square}(a, b)]$.

Definition 3 (mutation score)[13] Quality measurement of a test suite TC, which is the percentage of mutants detected, MS for short:

$$\text{MS}(\text{TC}) = \frac{M_k}{M_t - M_q}$$

where M_k is the number of mutants detected by the TC, M_t the total number of mutants, and M_q the number of equivalent mutants that cannot be detected by any set of test data.

Definition 4 (fault detection ratio)[13] The percentage of test cases that can detect certain mutant m, FD for short:

$$\text{FD}(m, \text{TC}) = \frac{N_f}{N_t - N_e}$$

where N_f is the number of times a program under a test fails, N_t the number of tests, and N_e the number of infeasible tests.

2 Case Studies

SpMatMul is a program for calculating multiplication of two sparse matrices, and TriSquare for computing triangle squares. MT is suitable for both of them.

2.1 Sparse matrix multiplication

The first case study is on SpMatMul which calculates the multiplication of two sparse matrices, whose code is as follows:

```
1  void SpMatMul(int n, int m,
2          const double *a, const int *ia, const int *ja,
3          const double *b, const int *ib, const int *jb,
4          double *c, int *ic, int *jc){
5  int nz=0, *mask, i, j, k, icol, icol_add;
6     const double *aij=a;
7     const int *neighbour=ja;
8
9  mask=(int *)malloc(m * sizeof(int));
10 for(i=0; i<m; i++) mask[i]=-1;
11 ic[0]=0;
12 for(i=0; i<n; i++){
13   for(j=ia[i]; j<ia[i+1]; j++){
14    for(k=ib[*neighbour]; k<ib[(*neighbour)+1]; k++){
15      icol_add=jb[k];
16      icol=mask[icol_add];
17      if(icol==-1){
18      jc[nz]=icol_add;
19      c[nz]=(*aij) * b[k];
20      mask[icol_add]=nz;
21      nz++;
22      }
23      else
24      c[icol] +=(*aij) * b[k];
25     }
26     aij++;
27     neighbour++;
28   }
29   for(j=ic[i]; j<nz; j++) mask[jc[j]]=-1;
30  ic[i+1]=nz;
31 }
32 free(mask);
33 }
```

This program has been described detailedly in Ref. [13], and five mutants[13] are designed for it:

Mutant 1: Delete line 19;

Mutant 2: Replace line 12 with "$c[nz] = (*aij)$";

Mutant 3: Replace line 12 with "$c[nz] = b[k]$";

Mutant 4: Replace line 17 with "$c[icol] += (*aij)$";

Mutant 5: Replace line 17 with "$c[icol] += b[k]$".

Here, we only consider nine MRs (MR_1 to MR_9 in Tab. 1) and eight special cases (SC_1 to SC_8 in Tab. 2) as OTCs.

Tab. 1 Nine MRs for SpMatMul[13]

MR_i	R	R_f
MR_1	$\boldsymbol{A}' = \boldsymbol{B}^{\mathrm{T}}, \boldsymbol{B}' = \boldsymbol{A}^{\mathrm{T}}$	$\boldsymbol{A}'\boldsymbol{B}' = (\boldsymbol{AB})^{\mathrm{T}}$
MR_2	$\boldsymbol{A}' = \boldsymbol{PA}, \boldsymbol{B}' = \boldsymbol{B}$	$\boldsymbol{A}'\boldsymbol{B}' = \boldsymbol{P}(\boldsymbol{AB})$
MR_3	$\boldsymbol{A}' = \boldsymbol{A}, \boldsymbol{B}' = \boldsymbol{PB}$	$\boldsymbol{A}'\boldsymbol{B}' = (\boldsymbol{AB})\boldsymbol{P}$
MR_4	$\boldsymbol{A}' = \boldsymbol{QA}, \boldsymbol{B}' = \boldsymbol{B}$	$\boldsymbol{A}'\boldsymbol{B}' = \boldsymbol{Q}(\boldsymbol{AB})$
MR_5	$\boldsymbol{A}' = \boldsymbol{A}, \boldsymbol{B}' = \boldsymbol{QB}$	$\boldsymbol{A}'\boldsymbol{B}' = (\boldsymbol{AB})\boldsymbol{Q}$
MR_6	$\boldsymbol{A}' = c\boldsymbol{A}, \boldsymbol{B}' = \boldsymbol{B}$	$\boldsymbol{A}'\boldsymbol{B}' = c(\boldsymbol{AB})$
MR_7	$\boldsymbol{A}' = \boldsymbol{A}, \boldsymbol{B}' = c\boldsymbol{B}$	$\boldsymbol{A}'\boldsymbol{B}' = c(\boldsymbol{AB})$
MR_8	$\boldsymbol{A}' = \boldsymbol{A} + \boldsymbol{I}, \boldsymbol{B}' = \boldsymbol{B}$	$\boldsymbol{A}'\boldsymbol{B}' = \boldsymbol{AB} + \boldsymbol{IB}$
MR_9	$\boldsymbol{A}' = \boldsymbol{A}, \boldsymbol{B}' = \boldsymbol{B} + \boldsymbol{I}$	$\boldsymbol{A}'\boldsymbol{B}' = \boldsymbol{AI} + \boldsymbol{AB}$

Notes: $\boldsymbol{P}$ is obtained by exchanging two rows of the identity matrix $\boldsymbol{I}$. $\boldsymbol{Q}$ is obtained by multiplying a principal diagonal element $\boldsymbol{I}$ with a scalar a. c is a scalar.

Tab. 2 Special cases for SpMatMul[13]

SC_i	$\boldsymbol{A} = [a_{ij}]_{n \times r}, \boldsymbol{B} = [b_{ij}]_{r \times m}$	$\boldsymbol{C} = \boldsymbol{AB} = [c_{ij}]_{n \times m}$
SC_1	$\boldsymbol{A} = \boldsymbol{0}$	$\boldsymbol{0}$
SC_2	$\boldsymbol{B} = \boldsymbol{0}$	$\boldsymbol{0}$
SC_3	$\boldsymbol{A} = \boldsymbol{I}$	$\boldsymbol{B}$
SC_4	$\boldsymbol{B} = \boldsymbol{I}$	$\boldsymbol{A}$
SC_5	$n = 1, a_{ij} = \begin{cases} 1 & i = j = 1 \\ 0 & \text{otherwise} \end{cases}$	$c_{ij} = b_{nj}$
SC_6	$m = 1, b_{ij} = \begin{cases} 1 & i = j = 1 \\ 0 & \text{otherwise} \end{cases}$	$c_{ij} = a_{im}$
SC_7	$r = 1, a_{ij} = 1$	$c_{ij} = b_{rj}$
SC_8	$r = 1, b_{ij} = 1$	$c_{ij} = a_{ir}$

As Tab. 3 in Ref. [13] shows: 1) Mutant 4 and 5 cannot

be detected no matter what MR is used; 2) Most of the fault detection ratio (FD) values for mutant 1, 2 and 3 are not more than 75%; 3) None of the five mutants can be discovered with MR_3. These results indicate that testing SpMatMul with SC_{1-8} and MR_{1-9} is not effective, and some test cases are useless.

2.2 Triangle square calculation

The second case study is TriSquare. Its code is listed as follows:

```
double TriSquare(int a, int b, int c) {
1   int match = 0;
P1  if(a = = b)
2       match = match + 1;
P2  if(a = = c)
3       match = match + 2;
P3  if(b = = c)
4       match = match + 3;
P4  if(match = = 0) {/* if a, b and c are not equal to each other */
P5      if(a + b < = c) {
5           System.out.println("Not a triangle");
6           return 0.0;
P6      } else if(b + c < = a) {
7           System.out.println("Not a triangle");
8           return 0.0;
P7      } else if(a + c < = b) {
9           System.out.println("Not a triangle");
10          return 0.0;
        } else {
11          double p = (a + b + c)/2.0;
12          System.out.println("Scalene");
13          return sqrt(p * (p - a) * (p - b) * (p - c)); /* compute square */
        }
P8  } else if(match = = 1) {/* if(a = b ≠ c) */
P9      if(a + b < = c) {
14          System.out.println("Not a triangle");
15          return 0.0;
        } else {
16          double h = sqrt(pow(a, 2) - pow(c/2.0, 2));
17          System.out.println("Isosceles");
18          return(c * h)/2.0; /* compute square */
        }
P10 } else if(match = = 2) {/* if(a = c ≠ b) */
P11     if(a + c < = b) {
19          System.out.println("Not a triangle");
20          return 0.0;
        } else {
21          double h = sqrt(pow(a, 2) - pow(b/2.0, 2));
22          System.out.println("Isosceles");
23          return(b * h)/2.0; /* compute square */
        }
P12 } else if(match = = 3) {/* if(b = c ≠ a) */
P13     if(b + c < = a) {
24          System.out.println("Not a triangle.");
25          return 0.0;
        } else {
26          double h = sqrt(pow(b, 2) - pow(a/2.0, 2));
27          System.out.println("Isosceles");
28          return(a * h)/2.0; /* compute square */
        }
    } else {/* if(a = b = c) */
29      System.out.println("Equilateral");
30      return(sqrt(3.0) * a * a)/4.0; /* compute square */
    }
}
```

This program first decides whether three positive real numbers, a, b and c can form a triangle. If so, type and square of this triangle are calculated.

We construct seven MRs for TriSquare, and details are shown in Tab. 3. Among them, mr_5 is constructed based on parallelogram characteristics. In Fig. 1, it is obvious that $OD = OB = b$, so the squares of triangle ODC and OBC are equivalent to each other; that is, TriSquare(a, b, c) = TriSquare(k, b, c). Because $|BD|^2 + |AC|^2 = |AB|^2 + |BC|^2 + |CD|^2 + |DA|^2$, $k = \sqrt{2b^2 + 2c^2 - a^2}$. $mr_{6,7}$ are constructed in the same way.

Tab. 3 Seven MRs for TriSquare

MR	r	r_f
mr_1	$(a' = b, b' = a, c' = c)$	
mr_2	$(a' = a, b' = c, c' = b)$	For $mr_{1,2,3,5,6,7}$, TriSquare(a', b', c') = TriSquare(a, b, c);
mr_3	$(a' = c, b' = b, c' = a)$	
mr_4	$(a' = 2a, b' = 2b, c' = 2c)$	For mr_4, TriSquare(a', b', c') = 4 × TriSquare(a, b, c)
mr_5	$(a' = \sqrt{2b^2 + 2c^2 - a^2}, b' = b, c' = c)$	
mr_6	$(a' = a, b' = \sqrt{2a^2 + 2c^2 - b^2}, c' = c)$	
mr_7	$(a' = a, b' = b, c' = \sqrt{2a^2 + 2b^2 - c^2})$	

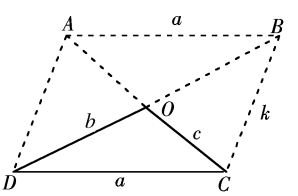

Fig. 1 Principle of mr_5

Obviously, when a, b and c can form a triangle, that is, when Gres is true, where Gres = $(a > 0) \wedge (b > 0) \wedge (c > 0) \wedge (a + b > c) \wedge (a + c > b) \wedge (b + c > a)$, it is effective to test TriSquare with MT. And test cases for it can be divided into five categories using equivalent partition, as Tab. 4 illustrates.

Tab. 4 Five case types for TriSquare

Type	Restrictions
Tri1	$(a \neq b \neq c \neq a) \wedge$ Gres
Tri2	$(a = b \neq c) \wedge$ Gres
Tri3	$(a = c \neq b) \wedge$ Gres
Tri4	$(b = c \neq a) \wedge$ Gres
Tri5	$a = b = c > 0$

Here, we also use a mutation analysis technique[15] to estimate test suites and MRs. Four mutants are imported based on two types of mutation operators, AOR (arithmetic operator replacement) and DSA (data statement alterations)[15]:

Mutant 1: Exchange sentence 2 and 3;

Mutant 2: Replace sentence 11 with "$p = (a + b + c) * 2$";

Mutant 3: Replace "/2" in 18, 23 and 28 with "$*2$";

Mutant 4: Replace sentence 30 with "return sqrt(3) $* a * a/2$".

Testing results are given in Tab. 5. Numbers in the second column represent the mutants that corresponding types of inputs can arouse. For example, the number "2" in the first row indicates that the sentence including mutant 2 will be executed when TriSquare is run with an input of type Tri1. Other numbers denote the mutants that relevant types of OTC and MRs can detect. Obviously, $mr_{1,3}$ can only discover mutant 1, while $mr_{2,4}$ can discover none. But $mr_{5,6,7}$ perform better, and their FTCs can also detect mutants.

Tab. 5 Testing results for TriSquare

Tri	Mutant	mr_1	mr_2	mr_3	mr_4	mr_5	mr_6	mr_7
Tri1	2					2	2	2
Tri2	1, 3			1		1, 2, 3	1, 2, 3	
Tri3	1, 3	1				1, 2, 3		1, 2, 3
Tri4	3	1		1			2, 3	2, 3
Tri5	4					3, 4	3, 4	3, 4

3 Testing with Compositional MRs

Although MT is effective in solving oracle problems, some MRs with poor performance are often built during testing, such as $mr_{2,4}$ in case study 2. In this section, we discuss the construction of CMRs to improve testing efficiency.

3.1 CMRs for SpMatMul

For SpMatMul, MR_2 and MR_3, MR_4 and MR_5, MR_6 and MR_7, and MR_8 and MR_9 are symmetric[13], so we divide the 9 MRs into 5 groups, MR_1, $MR_{2,3}$, $MR_{4,5}$, $MR_{6,7}$ and $MR_{8,9}$. In each of the last four groups of MRs, the first relation makes some primary transformation on matrix $\boldsymbol{A}$, while the second does so on matrix $\boldsymbol{B}$. We build CMRs for SpMatMul on three angles: 1) A new CMR is composed of MRs selected from each group, and it makes one matrix of the OTC change as great as possible, and the other as small as possible; 2) A new CMR is composed of MRs selected from each group, and it makes the OTC's two matrices change to a similar extent; 3) A new CMR can make some of its FTCs from SC_{1-8} execute the sentence including mutants 4 and 5 (line 17 in code).

As to 3), we should analyze the structure of SpMatMul first. Suppose matrix $\boldsymbol{C} = [c_{ij}]_{n\times m}$ is the multiplication of two matrices, $\boldsymbol{A} = [a_{ij}]_{n\times r}$ and $\boldsymbol{B} = [b_{ij}]_{r\times m}$. Each element of $\boldsymbol{C}$, $c_{i'j'}$, is obtained by multiplying a row of $\boldsymbol{A}$, $\boldsymbol{a} = \{a_{i'1}, a_{i'2}, \ldots, a_{i'r}\}$ and a column of $\boldsymbol{B}$, $\boldsymbol{b} = \{b_{1j'}, b_{2j'}, \ldots, b_{rj'}\}^{\mathrm{T}}$. A non-zero pair, $(a_{i'x}, b_{xj'})$ is a pair of elements, and neither $a_{i'x}$ nor $b_{xj'}$ is 0. If there are at least two non-zero pairs in $\boldsymbol{a}$ and $\boldsymbol{b}$, the sentence including mutant 4 and mutant 5 will be executed, and these two mutants might be detected. Testing with SC_{1-8} and their FTCs upon MR_{1-9} does not satisfy this demand.

Six CMRs ($CMRs_{1-6}$) are constructed for SpMatMul, as Tab. 6 shows. When these relations are mentioned, $CMRs_{1,2}$ respond to angle 1), $CMRs_{3,4}$ to angle 2), and $CMRs_{5,6}$ to angle 3). The second column of Tab. 6 gives the composition order of selected MRs.

Tab. 6 CMRs for SpMatMul

CMRs	MRs	r	r_f
$CMRs_1$	1, 2, 4, 6, 8	$\boldsymbol{A'} = \boldsymbol{B}^{\mathrm{T}}\boldsymbol{B'} = c[\boldsymbol{PQ}(\boldsymbol{A} + \boldsymbol{I})]^{\mathrm{T}}$	$\boldsymbol{A'B'} = c[(\boldsymbol{AB})^{\mathrm{T}} + (\boldsymbol{IB})^{\mathrm{T}}]\boldsymbol{QP}$
$CMRs_2$	1, 3, 5, 7, 9	$\boldsymbol{A'} = c[(\boldsymbol{B} + \boldsymbol{I})\boldsymbol{QP}]^{\mathrm{T}}\ \boldsymbol{B'} = \boldsymbol{A}^{\mathrm{T}}$	$\boldsymbol{A'B'} = c\boldsymbol{PQ}[(\boldsymbol{AB})^{\mathrm{T}} + (\boldsymbol{AI})^{\mathrm{T}}]$
$CMRs_3$	1, 2, 5, 6, 9	$\boldsymbol{A'} = \boldsymbol{Q}(\boldsymbol{B} + \boldsymbol{I})^{\mathrm{T}}\ \boldsymbol{B'} = c\boldsymbol{A}^{\mathrm{T}}\boldsymbol{P}$	$\boldsymbol{A'B'} = c\boldsymbol{Q}[(\boldsymbol{AB})^{\mathrm{T}} + (\boldsymbol{AI})^{\mathrm{T}}]\boldsymbol{P}$
$CMRs_4$	1, 3, 4, 7, 8	$\boldsymbol{A'} = c\boldsymbol{PB}^{\mathrm{T}}\ \boldsymbol{B'} = (\boldsymbol{A} + \boldsymbol{I})^{\mathrm{T}}\boldsymbol{Q}$	$\boldsymbol{A'B'} = c\boldsymbol{P}[(\boldsymbol{AB})^{\mathrm{T}} + (\boldsymbol{IB})^{\mathrm{T}}]\boldsymbol{Q}$
$CMRs_5$	1, 8, 2, 4	$\boldsymbol{A'} = \boldsymbol{B}^{\mathrm{T}}\ \boldsymbol{B'} = (\boldsymbol{PQA} + \boldsymbol{I})^{\mathrm{T}}$	$\boldsymbol{A'B'} = (\boldsymbol{AB})^{\mathrm{T}}\boldsymbol{QP} + (\boldsymbol{IB})^{\mathrm{T}}$
$CMRs_6$	1, 9, 3, 5	$\boldsymbol{A'} = (\boldsymbol{BQP} + \boldsymbol{I})^{\mathrm{T}}\ \boldsymbol{B'} = \boldsymbol{A}^{\mathrm{T}}$	$\boldsymbol{A'B'} = \boldsymbol{PQ}(\boldsymbol{AB})^{\mathrm{T}} + (\boldsymbol{AI})^{\mathrm{T}}$

These new CMRs are formed based on the technique of equivalent transformation. For example, $CMRs_1$ is composed of $MR_{1,2,4,6,8}$, which are

$$\mathrm{MR}_1 \quad R: \boldsymbol{A}^{(5)} = (\boldsymbol{B}^{(4)})^{\mathrm{T}}\ \boldsymbol{B}^{(5)} = (\boldsymbol{A}^{(4)})^{\mathrm{T}}$$
$$R_f: \boldsymbol{A}^{(5)}\boldsymbol{B}^{(5)} = (\boldsymbol{A}^{(4)}\boldsymbol{B}^{(4)})^{\mathrm{T}}$$

$$\mathrm{MR}_2 \quad R: \boldsymbol{A}^{(4)} = \boldsymbol{P}\boldsymbol{A}^{(3)}\ \boldsymbol{B}^{(4)} = \boldsymbol{B}^{(3)}$$
$$R_f: \boldsymbol{A}^{(4)}\boldsymbol{B}^{(4)} = \boldsymbol{P}(\boldsymbol{A}^{(3)}\boldsymbol{B}^{(3)})$$

$$\mathrm{MR}_4 \quad R: \boldsymbol{A}^{(3)} = \boldsymbol{Q}\boldsymbol{A}^{(2)}\ \boldsymbol{B}^{(3)} = \boldsymbol{B}^{(2)}$$
$$R_f: \boldsymbol{A}^{(3)}\boldsymbol{B}^{(3)} = \boldsymbol{Q}(\boldsymbol{A}^{(2)}\boldsymbol{B}^{(2)})$$

$$\mathrm{MR}_6 \quad R: \boldsymbol{A}^{(2)} = c\boldsymbol{A}^{(1)}\ \boldsymbol{B}^{(2)} = \boldsymbol{A}^{(1)}$$
$$R_f: \boldsymbol{A}^{(2)}\boldsymbol{B}^{(2)} = c(\boldsymbol{A}^{(1)}\boldsymbol{B}^{(1)})$$

$$\mathrm{MR}_8 \quad R: \boldsymbol{A}^{(1)} = \boldsymbol{A} + \boldsymbol{I}\ \boldsymbol{B}^{(1)} = \boldsymbol{B}$$
$$R_f: \boldsymbol{A}^{(1)}\boldsymbol{B}^{(1)} = \boldsymbol{AB} + \boldsymbol{IB}$$

We do pairwise composition on them in turn. First, $\boldsymbol{A}^{(4)}$ and $\boldsymbol{B}^{(4)}$ in MR_1 are replaced by the equivalent expression of $\boldsymbol{A}^{(3)}$ and $\boldsymbol{B}^{(3)}$ in MR_2, and we obtain a temporary CMR:

$$\mathrm{CMR}_x \quad R: \boldsymbol{A}^{(5)} = (\boldsymbol{B}^{(3)})^{\mathrm{T}}\boldsymbol{B}^{(5)} = (\boldsymbol{P}\boldsymbol{A}^{(3)})^{\mathrm{T}}$$
$$R_f: \boldsymbol{A}^{(5)}\boldsymbol{B}^{(5)} = [\boldsymbol{P}(\boldsymbol{A}^{(3)}\boldsymbol{B}^{(3)})]^{\mathrm{T}}$$

Then, CMR_x and MR_4 are composed. After four times of composition, $CMRs_1$ is built. Because SC_5 is not applicable for MR_2, $\boldsymbol{A}^{(3)}$ and $\boldsymbol{B}^{(3)}$ should not be this kind of matrices. Among SC_{1-8}, SC_5 is not applicable for $CMRs_1$.

To obtain the general trait of $CMRs_{1-6}$, we test each program with a mutant 20 times, and use $CMRs_{1-6}$ and SC_{1-8} each time. The results are given in Tab. 7. "m/t" at the cross of SC_i and $CMRs_j$ means that in 20 testings with SC_i and $CMRs_j$, mutant m can be discovered t times, and "—" means that SC_i is not applicable for $CMRs_j$.

We use the measurement criteria, mutation score (MS) and fault detection ratio (FD) to estimate the test suite here. As Tab. 8 shows, almost all FD values for mutants 1, 2, 3 are 75%, and the MS value for $CMRs_{5,6}$ is 1, that is, $CMRs_{5,6}$ can find all 5 mutants with SC_{1-8}. On the other hand, mutant1 cannot be detected by SC_1 upon MR_{1-9} (shown in Tab. 3(a) in Ref. [13]), but it can upon $CMRs_{1,4,5}$. All the facts indicate that $CMRs_{1-6}$ are more effective than MR_{1-9}.

Tab. 7 Testing results for SpMatMul with $CMRs_{1-6}$

SC	$CMRs_1$	$CMRs_2$	$CMRs_3$	$CMRs_4$	$CMRs_5$	$CMRs_6$
SC_1	1/20 2/20 3/20			1/20 2/16 3/20	1/20 3/20	
SC_2		1/19 2/20 3/20	1/15 2/20 3/17			2/20
SC_3	1/20 2/20 3/20	1/20 2/20 3/20	1/20 2/20 3/20	1/20 2/20 3/20	1/20 2/20 3/20 4/6 5/17	1/20 2/20 3/20
SC_4	1/20 2/20 3/20	1/20 2/20 3/20	1/20 2/20 3/20	1/20 2/20 3/20	1/20 2/20 3/20	1/20 2/20 3/20 4/17 5/7
SC_5	—	1/17 2/20 3/20	—	1/18 2/20 3/20	—	1/18 2/20 3/20
SC_6	1/14 2/20 3/18	—	1/16 2/16 3/19	—	1/14 2/19 3/16	—
SC_7	1/11 2/20 3/20	1/18 2/20 3/20	1/15 2/20 3/20	1/10 2/20 3/20	1/12 2/20 3/20	1/17 2/20 3/20
SC_8	1/20 2/20 3/20	1/20 2/20 3/20	1/20 2/20 3/20	1/20 2/20 3/20	1/20 2/20 3/20	1/20 2/20 3/20

Tab. 8 MS and FD values of testing with $CMRs_{1-6}$

CMRs	Mutant/%					M_k	MS(T)
	1	2	3	4	5		
$CMRs_1$	75	75	75	0	0	3	0.6
$CMRs_2$	75	75	75	0	0	3	0.6
$CMRs_3$	75	75	75	0	0	3	0.6
$CMRs_4$	75	75	75	0	0	3	0.6
$CMRs_5$	75	62.5	75	25	25	5	1
$CMRs_6$	62.5	75	62.5	25	25	5	1

3.2 CMRs for TriSquare

According to their structural characteristics, TriSquare's seven MRs can be partitioned into three groups: $mr_{1,2,3}$, $mr_{5,6,7}$ and mr_4. When constructing CMRs, we focus on two aspects:

1) New CMR is composed by MRs selected from each group, and more than one MR is chosen from $mr_{1,2,3}$;

2) New CMR is composed by MRs selected from each group, and more than one MR is chosen from $mr_{4,5,6}$.

We also form six CMRs ($CMRt_{1-6}$) for TriSquare, as Tab. 9 shows. Among them, $CMRt_{1,2,3}$ respond to aspect 1), while $CMRt_{4,5,6}$ respond to aspect 2).

Tab. 9 CMRs for TriSquare

CMRt	MRs	r	r_f
$CMRt_1$	4, 2, 1, 5	$(a'=2b, b'=2c, c'=2\sqrt{2b^2+2c^2-a^2})$	TriSquare (a', b', c') $=4\times$ TriSquare (a, b, c)
$CMRt_2$	4, 3, 2, 6	$(a'=2\sqrt{2a^2+2c^2-b^2}, b'=2c, c'=2a)$	
$CMRt_3$	4, 1, 3, 7	$(a'=2b, b'=2\sqrt{2a^2+2b^2-c^2}, c'=2a)$	
$CMRt_4$	4, 3, 6, 5	$(a'=2c, b'=2\sqrt{3b^2+6c^2-2a^2}, c'=2\sqrt{2b^2+2c^2-a^2})$	
$CMRt_5$	4, 1, 7, 6	$(a'=2\sqrt{2a^2+2c^2-b^2}, b'=2a, c'=2\sqrt{3c^2+6a^2-2b^2})$	
$CMRt_6$	4, 2, 5, 7	$(a'=2\sqrt{3a^2+6b^2-2c^2}, b'=2\sqrt{2a^2+2b^2-c^2}, c'=2b)$	

Testing results are illustrated in Fig. 2. The y-axis of this figure represents four mutants of TriSquare, and the x-axis indicates testing with corresponding types of the OTC(number on x-axis ($Tri_{1\text{-}5}$)) and CMR($CMRt_{1-6}$). Here, an input I is said to arouse a mutant M if the run of TriSquare with I can make the sentence including M be executed. A gray rectangle denotes that the mutant aroused by a relevant type of the OTC can be detected by relevant CMR. A shaded rectangle denotes the detection of a mutant that is not aroused by a relevant type of OTC(aroused by FTC). A black rectangle denotes that the mutant aroused by a relevant type of OTC cannot be detected.

As Fig. 2 shows, all four mutants could be found by any of these six CMRs. Compared to mr_{1-7}, $CMRt_{1-6}$ perform perfectly in testing TriSquare. When $CMRt_{1-6}$ themselves are mentioned, $CMRt_{4,5,6}$ are better than $CMRt_{1,2,3}$, because the former can detect any mutants that are aroused by OTCs. The experimental results show that the greater an MR makes its OTC and FTC different, the better it is. For example, sometimes both the OTC and the FTC of $CMRt_{1,2,3}$ are isoceles triangles, and they execute the program with the same data, so mutants cannot be detected. But this case will not take place when $CMRt_{4,5,6}$ are used, because in

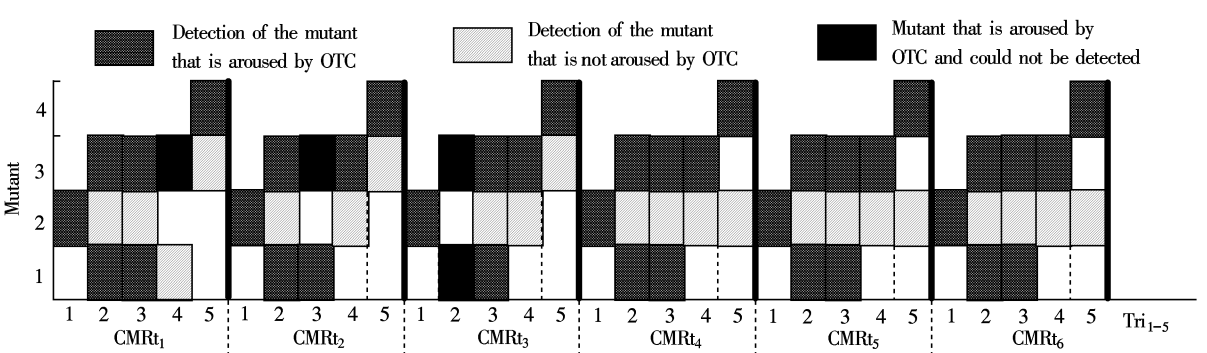

Fig. 2 Testing results for TriSquare with $CMRt_{1-6}$

most cases their FTCs are anomalistic triangles which are much different from the OTCs.

3.3 Conclusion for experiment

Based on the above data, the following conclusions can be drawn:

1) The application domain of the CMR should be clearly defined. That is to say, when mr_1, mr_2, …, mr_i, mr_{i+1}, …, and mr_n are composed, the FTC of mr_{i+1} should be used as the OTC for mr_i. Otherwise, the CMR is illegal itself, and the testing results are not believable. As for the first case study, SC_5 is not applicable for MR_2, and this unconformity should be embodied by the limitation on a new CMR's OTC, in other words, the application domain of a new CMR should be strictly prescribed.

2) MR selection greatly affects the performance of a new CMR. In fact, the capability of a CMR is mostly decided by several central MRs forming it. For the case study of TriSquare, the central MRs of $CMRt_4$ and $CMRt_1$ are $mr_{5,6}$ and mr_5, respectively, so their testing capabilities are different from each other.

3) Sometimes, the sequence of composition has a great effect on a new CMR's efficiency. For example, in case study 1, when MR_8 is composed before MR_2, mutants 4 and 5 will be detected by SC_3. Otherwise, they cannot be found. So $CMRs_1$ and $CMRs_5$ perform diversely. This is mostly because the switch of the composition sequence changes the essence of the testing. But this case is not always true. The CMRs will not have a bit of difference; however (r_1, r_{f1}) and (r_2, r_{f2}) are composed.

4) New CMR should make the difference between its OTC and FTC much larger than those of each MR forming it. The difference is embodied in multi-aspects. First, as for the program structure, $CMRs_{5,6}$ of SpMatMul can generate FTCs running a different program path for $SC_{3,4}$, but MR_{1-9} cannot; Secondly, as for the program function, $CMRt_{4,5,6}$ of TriSquare can generate FTCs with entirely different shapes for isosceles triangle OTCs, but mr_{1-4} cannot. So the CMRs of these two programs are much more effective than their originally formed MRs.

5) With CMRs, errors can be detected by much fewer test cases. A lot of redundancy is avoided when CMRs are used, and the error detection performance of CMRs is greatly improved, so we can find errors with much fewer test cases.

4 Conclusion

In this paper, testing on CMR is researched systematically. We first present the concept of CMR based on the speculative law of proposition logic, and then prove its usefulness in error detection with two case studies. Finally, some conclusions for CMR construction are provided.

As future work, we would like to provide algorithms and tools for constructing CMR automatically.

References

[1] Weyuker E J. On testing non-testable programs [J]. *The Computer Journal*, 1982, **25**(4): 465 − 470.

[2] Chen T Y, Cheung S C, Yiu S M. Metamorphic testing: a new approach for generating next test cases [R]. Hong Kong: Department of Computer Science of Hong Kong University, 1998.

[3] Chen T Y, Feng J, Tse T H. Metamorphic testing of programs on partial differential equations: a case study [C]// *Proc of the* 26*th Annual International Computer Software and Applications Conference*. Washington DC: IEEE Computer Society, 2002: 327 − 333.

[4] Gotlieb A, Botella B. Automated metamorphic testing [C]// *Proc of the* 27*th Annual International Computer Software and Applications Conference*. Washington DC: IEEE Computer Society, 2003: 34.

[5] Zhou Z Q, Huang D H, Tse T H, et al. Metamorphic testing and its applications [C]// *Proc of the* 8*th International Symposium on Future Software Technology*. Washington DC: IEEE Computer Society, 2004: 23 − 28.

[6] Tse T H, Yau S S, Chan W K, et al. Testing context-sensitive middleware-based software applications [C]// *Proc of the* 28*th Annual International Computer Software and Applications Conference*. Washington DC: IEEE Computer Society, 2004: 458 − 466.

[7] Chan W K, Chen T Y, Heng L, et al. A metamorphic approach to integration testing of context-sensitive middleware-based applications [C]// *Proc of the* 5*th Annual International Conference on Quality Software*. Washington DC: IEEE Computer Society, 2005: 241 − 249.

[8] Chan W K, Cheung S C, Leung K P H. Towards a metamorphic testing methodology for service-oriented software applications [C]// *Proc of the* 1*st International Conference on Services Engineering*. Washington DC: IEEE Computer Society, 2005: 470 − 476.

[9] Chan W K, Cheung S C, Leung R P H. A metamorphic testing approach for online testing of service-oriented software applications [J]. *International Journal of Web Services Research*, 2007, **2**(1): 60 − 80.

[10] Chen T Y, Huang D H, Tse T H, et al. Case studies on the selection of useful relations in metamorphic testing [R]. Hong Kong: Department of Computer Science of Hong Kong University, 1998.

[11] Chen T Y, Tse T H, Zhou Z Q. Semi-proving: an integrated method based on global symbolic evaluation and metamorphic testing [J]. *ACM SIGSOFT Software Engineering Notes*, 2002, **27**(4): 191 − 195.

[12] Chen T Y, Tse T H, Zhou Z Q. Fault-based testing without the need of oracles [J]. *Information and Software Technology*, 2003, **45**(1): 1 − 9.

[13] Wu P, Shi X C, Tang J J, et al. Metamorphic testing and special case testing: a case study [J]. *Journal of Software*, 2005, **16**(7): 1210 − 1220.

[14] Wu P. Iterative metamorphic testing [C]// *Proc of the* 29*th Annual International Computer Software and Applications Conference*. Washington DC: IEEE Computer Society, 2005: 19 − 24.

[15] Offutt A J, Lee A, Rothermel G, et al. An experimental determination of sufficient mutant operators [J]. *ACM Transactions on Software Engineering and Methodology*, 1996, **5**(2): 99 − 118.

使用复合蜕变关系进行软件测试的实例研究

董国伟　徐宝文　陈　林　聂长海　王璐璐

（东南大学计算机科学与工程学院，南京 210096）
（江苏省软件质量研究所，南京 210096）

摘要：蜕变测试时经常会出现蜕变关系检错能力低下的情况. 基于命题逻辑的推理规则，提出了复合蜕变关系的构造方法，该方法对已构造的关系依次进行两两复合最终得到新的蜕变关系. 复合蜕变关系可以把原关系的优点综合起来，具有更强的检错能力. 此外，由于将蜕变关系复合后关系数量减少，所以当使用它测试程序时，生成测试用例的数量会大幅度降低. 通过 2 个实例对复合蜕变关系的测试性能进行研究，实验结果表明复合关系的性能主要取决于构成它的核心蜕变关系，以及关系复合的顺序. 使用复合蜕变关系可以极大地提高测试效率.

关键词：软件测试；蜕变测试；复合蜕变关系
中图分类号：TP311