Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 14, no. 7, July 2018, pp. 1481-1486
DOI: 10.23940/ijpe.18.07.p11.14811486

Metamorphic Testing for Oracle Problem in Integer Bug Detection

Yi Yao and Jialuo Liu*

Command and Control Engineering College, Army Engineering University of PLA, Nanjing, 210007, China

Abstract

Integer defects are an important cause of software quality degradation. An explicit expected output plays an important role in the traditional theory of software testing, but it is very difficult for much software to get the expected output since ascertaining the validity of the actual output is very hard. Integer bugs are always ignored because of the Test Oracle problem. A metamorphic relationship that can find out the potential error is presented. The experimental results show that the mean of integer bugs detection based on the metamorphosis relation can detect the invisible unexpected output, which is unable to get in traditional means. In addition, the effectiveness of detecting integer defects is improved.

Keywords: metamorphic relationship; test oracle; integer bugs

(Submitted on March 25, 2018; Revised on May 11, 2018; Accepted on June 25, 2018)

1. Introduction

Software testing is a process of auditing or comparing the real output with anticipated output to judge if the test is successful. However, there are two basic problems in software testing: the reliability test case set problem and 'Test Oracle' problem. The premise of software testing theory needs a definite anticipated-output to judge whether the examination passes. But it is very difficult for much software to get the expected output as ascertaining the validity of the real output is very hard. There is the 'non-testable program' that is impacted on the nature of the problem itself rather than ascribing to human factors (such as software testing is short of the pre-requisite process file, etc.) [2]. This paper has made improvements based on the paper of Research on Metamorphic Testing [1].

In order to solve the problem of test determination, University of Hong Kong Professor Tsong Yueh Chen [3,4,5,6,7] came up with the Metamorphic Testing (MT) technology in 1998. The test Oracle issue can be partly solved by taking advantage of the metamorphic relationship and the initial test case to create the additional test case. It initially judged that there are errors in the software by verifying that the output of a separate multi-version program. In the practical test, because they always lack multi-versions program, technology has an ability to determine whether the output was consistent with the nature based on different inputs, as long as the program was designed to conform to nature. Then, whether the program satisfies this property can be determined, which ultimately judges the validity of the program to a certain extent.

As software testing becomes more important in software assurance, software testing is an important method to ensure high-quality software. 'Test Oracle' has turned into the most important problem that restricts the progress of software testing [8,9]. This problem is dominant in software security testing because it is harder to acquire the requirement of software security testing than other types, and it is hard to test what type of behavior of software is secure. It is also difficult to put forward the passing criteria of the security test. The progress of software security testing technology is seriously limited by the software security Test Oracle problem. Therefore, how to figure out the problem of the developing software security test without the criterion of safety test determination is of great research value. Metamorphic Testing was originally proposed to solve the test determination problem in the numerical calculation program, and was later extended to other types of applications.

* Corresponding author.
E-mail address: 17551037134@163.com

The integer bug is the primary cause that led to the software calculation fault. The integer variables are represented by the fixed-bit-wide vector in the computer program. An integer overflow occurs when the value obtained after an instruction operation is greater or less than the range represented by the bit size of the stored result. When the rocket of Ariane5 was initially launched, an integer overflow was aroused. A 64-bit floating point number was converted to a 16-bit signed integer. The control instructions were incorrectly issued by the control system of the rocket, which ultimately leads to the disastrous explosion of the rocket. In addition, if the result of every arithmetic operation is detected, the detection cost will be too high, so the integer overflow has not been examined in commercial software. For example, software security vulnerabilities are created if a program finds a non-expected value for an integer and uses it for array indexing or looping variables. In the preliminary version of this paper, a way to examine integer errors by using metamorphic relation is presented, and an example is given to verify the validity of the method. In this paper, the following two points have been improved: 1) This paper has some pertinence to the initial test input and the selection of metamorphic relationship, and 2) This paper compares traditional integer error testing methods. It is testified by experiments that this method detects invisible abnormal failures that cannot be detected by traditional testing techniques.

2. Ways of Integer Bugs Detection

At present, the integer error detection method mainly adopts three methods: precondition, error detection and postcondition. Among them, the precondition examines if a fault occurs before executing the command. For example, the priori condition for the shortest distance between two points is the line. This specifies that the method must be true before it can be called; then, the code is examined one by one using a static analysis tool based on this formal rule. Error detection demanded to judge if the faults occur during the execution of the operation. Because integer bugs are handled in limitations of apparatus, operating systems basiclly provide a positive treatment mechanism for overflow error. Postconditions are to execute operations initially; the result is drawn by comparing the test value with the anticipated value. It is the most commonly used examination method.

Although there are advantages and disadvantages of the three methods, for not directly get the expected results, the program output is reasonable. But, the incorrect integer error is not applicable so many testers find that the method of these three testers is generally difficult to efficiently and accurately conclude the test.

3. Means of Integer Bugs Detection based on Metamorphic Testing

For the purpose of solving the problem of "Test Oracle" in the existing the means of integer bugs detection, the integer errors detection mean based on the Metamorphic test is presented. The method is substantially based on verification of validity. When an integer error is discovered, the program must have found a false value for an integer. But if the false value is involved in the following operation, the program will ultimately either crash or count an unanticipated output. In other words, when the input of software satisfies specific properties, the homologous output of software also satisfies homologous properties.

Before describing the method in detail, formalized definitions of ideas used in the method is given.

Definition 1: It is supposed that program P is a realization of the function f. x1,x2,…,xn, (n>1) are n-group argument for function f, and f(x1),f(x2),…,f(xn) are the output of the corresponding function f. When x1, x2,..., xn satisfies the relationship r, and f(x1), f(x2),..., f(xn) satisfies the relationship rf, (r, rf) is named the metamorphic relationship of program P.

Definition 2: For the same program P, more than Metamorphic relationships (r, rf) need to be verified or can be extracted. Using the original test case, a new set of use cases is generated in combination with relational r, and the derived test cases are obtained.

The examination of integral bugs method based on the Metamorphic test consists of three steps:

Step 1: The initial test cases are chosen. For program P, choose I1, I2, … as inputs of program P homologous to x1, x2, ..., xn. The source test case (I1, I2, …, In) is obtained.

Step 2: The true metamorphic relationship is selected to produce derived test cases. We select the applicable metamorphic relationship R = (r, rf) from P. It is supposed that P is true, according to the Definition 1, $r(I1, I2, \ldots, In) \Longrightarrow rf\big(P(I1), P(I2), \ldots, P(In)\big)$. It means getting the derivative test cases. If there are multiple metamorphic relationships, you

can also select several transformation relationships R1,R2,...,Rn to generate multiple types of derivative test cases to enhance the correctness of the tests.

Step 3: The initial test case with the derived test cases is compared to determine if the metamorphic relationship is satisfied. If P is true, P meets $r(I1, I2, \ldots, In) \Longrightarrow rf(P(I1), P(I2), \ldots, P(In))$. If the test case cannot satisfy the formula above, the hypothesis does not hold and the program is false.

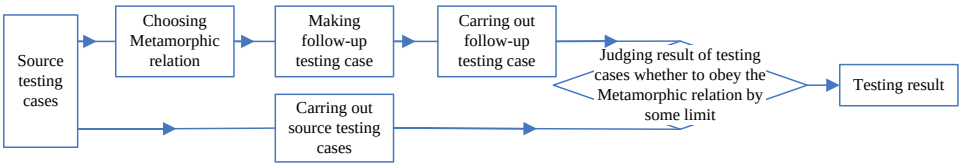

Figure 1. Process of integer bugs detection based on metamorphic testing

4. Case Study

Programs for counting graphics area on the rectangular coordinate system have been used to computer graphics. In this paper, a case study of the metamorphic testing for integer bugs detection is carried out by counting the graph area on the rectangular coordinate system.

The program is as follows: vertex coordinates of the convex polygon are input in the txt file. It is used as the input of the program to count the area and perimeter. The procedure of the area count is that convex polygon is divided into some triangles. Then, calculate the area of each triangle according to Helen's formula and sum it up.

4.1. Variation Design

The statement "p = (a + b + c) / 2" is modified as "p = (a + b + (int) c) / 2". Meanwhile, the variant version of the program that includes integer bugs is denoted as mutant1. Some minor human errors are injected into software. The procedure after the injection of a mutation is named a variant. The codes are displayed below.

```
double area
{   double a, b, c, p, area;
    a = side(v1, v2);
    b = side(v2, v3);
    c = side(v3, v1);
    p = (a + b + (int)c) / 2;// True version: p = (a + b + c) / 2;
    area = sqrt(p*(p - a)*(p - b)*(p - c));
    return area;   }
```

4.2. Metamorphic Relationship

The following metamorphic relationships can be obtained using the area and perimeter of similar triangles proportional to the length of the side, as shown in Figure 2.

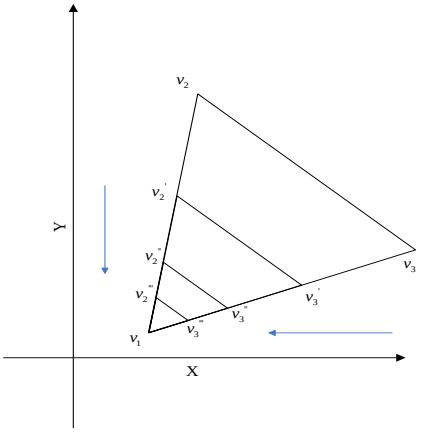

Figure 2. The area and perimeter of similar triangles are proportional to the length of the side

The metamorphic relationships are respectively marked as Flag1 (the ratio of the area) and Flag2 (the ratio of the perimeter). Then, several equations can be established as follows.

$$\begin{aligned}
&triangle(v_1,v_2,v_3), triangle(v_1,v_2^{'},v_3^{'}), triangle(v_1,v_2^{''},v_3^{''}), triangle(v_1,v_2^{'''},v_3^{'''})\\
&\text{and } \overrightarrow{v_1v_2}=2\overrightarrow{v_1v_2^{'}}=4\overrightarrow{v_1v_2^{''}}=8\overrightarrow{v_1v_2^{'''}}, \overrightarrow{v_1v_3}=2\overrightarrow{v_1v_3^{'}}=4\overrightarrow{v_1v_3^{''}}=8\overrightarrow{v_1v_3^{'''}}\\
&\Rightarrow\begin{cases}
\dfrac{perimeter(v_1,v_2,v_3)}{perimeter(v_1,v_2^{'},v_3^{'})}=\dfrac{perimeter(v_1,v_2^{'},v_3^{'})}{perimeter(v_1,v_2^{''},v_3^{''})}=\dfrac{perimeter(v_1,v_2^{''},v_3^{''})}{perimeter(v_1,v_2^{'''},v_3^{'''})}=2(\text{ratio of perimeter})\\
\dfrac{area(v_1,v_2,v_3)}{area(v_1,v_2^{'},v_3^{'})}=\dfrac{area(v_1,v_2^{'},v_3^{'})}{area(v_1,v_2^{''},v_3^{''})}=\dfrac{area(v_1,v_2^{''},v_3^{''})}{area(v_1,v_2^{'''},v_3^{'''})}=4(\text{ratio of area})
\end{cases}
\end{aligned} \tag{1}$$

4.3. Testing Cases Design

Considering that integer errors often happen near the boundary of the input domain, two sets of data within the interval [64534,66536] and [-66535, -64535] are randomly generated. 10 groups of the original test input are randomly produced by the two data set, as displayed in Table 1.

Table 1. Original test input

INPUT(NO.)	v_1		v_2		v_3	
	x1	y1	x2	y2	x3	y3
p1	65965.1	66369	65976.6	66089.3	65756	65818.9
p2	66394.3	65858.1	65877.5	65960.3	66280.4	66246
p3	-64787.6	65606	66360	65830.7	65740.6	-65138.1
p4	-65327.4	-66625.6	66154.9	-65363.1	-65462	-64682.6
p5	-65031	66381.5	66262.8	-64673.7	-64817.4	65958.3
p6	-64997.2	-65031.9	-65249.1	65726.8	66037	66370.1
p7	65920.9	65958.3	-64822	65936.1	66070.8	65996.4
p8	65675	66370.1	-65109.7	-64766.3	-64699	-65296.6
p9	-65312	65996.4	-64697.9	-65147.1	-65209.9	-65031.9
p10	-65363.1	-65296.6	-65071.6	-65393	-64686.5	-64615

Generate the derivative test input according to Flag1 and Flag2 as follows.

$$\begin{aligned}
&\frac{v_1+v_2}{2}=v_2^{'}, \frac{v_1+v_3}{2}=v_3^{'}; \quad \frac{v_1+v_2^{'}}{2}=v_2^{''}, \frac{v_1+v_3^{'}}{2}=v_3^{''};\\
&\frac{v_1+v_2^{''}}{2}=v_2^{'''}, \frac{v_1+v_3^{''}}{2}=v_3^{'''}
\end{aligned} \tag{2}$$

Because of errors in floating point calculation, it is necessary to convert the two formulas of transformation relation, Flag1 and Flag2, to eliminate the influence caused by these errors. The new formula is shown below.

$$\begin{cases}
\left|\dfrac{perimeter(Si)}{perimeter(Y_1i)}-2\right|=\sigma_1, \left|\dfrac{perimeter(Y_1i)}{perimeter(Y_2i)}-2\right|=\sigma_2, \left|\dfrac{perimeter(Y_2i)}{perimeter(Y_3i)}-2\right|=\sigma_3\\
\text{AND} \quad \left|\dfrac{area(Si)}{area(Y_1i)}-4\right|=\varepsilon_1, \left|\dfrac{area(Y_1i)}{area(Y_2i)}-4\right|=\varepsilon_2, \left|\dfrac{area(Y_2i)}{area(Y_3i)}-4\right|=\varepsilon_3\\
i\in[1,10], \quad \overline{\Delta s_1}=\dfrac{1}{3}\sum_{j=1}^{3}\sigma_j<10^{-3}, \overline{\Delta s_2}=\dfrac{1}{3}\sum_{j=1}^{3}\varepsilon_j<10^{-3}
\end{cases} \tag{3}$$

According to Flag1 and Flag2, Y1i, Y2i, Y3i are produced for three sets of derivative test inputs. When the conditions are satisfied, test cases (Si, Y1i, Y2i, Y3i) meet Flag1 and Flag2 of the metamorphic relationships.

4.4. Testing Result

Consequences are obtained as displayed in Table 2. Test case results through executing in program version before and after mutation are compared; the conclusion that test cases set {(Si,Y1i,Y2i,Y3i)| I=1,2,3,...,9,10} produced by Flag1 and Flag2 can test the output exception of the program by mutating. So, mutant1 is defective. Therefore, it can be proved that the integer bugs detection method based on metamorphic relationship is effective in detecting recessive integer bugs.

Table 2. Test result

Test Case	Source $\overline{\Delta s_1}, \overline{\Delta s_2}$	mutant1 $\overline{\Delta s_1}, \overline{\Delta s_2}$
（$S1$，$Y_1 1$，$Y_2 1$，$Y_3 1$）	$\overline{\Delta s_1} = 2.39013\times10^{-4}$, $\overline{\Delta s_2} = 7.11892\times10^{-5}$	$\boxed{\overline{\Delta s_1} = 8.49250\times10^{-2}}$, $\overline{\Delta s_2} = 7.11892\times10^{-5}$
（$S2$，$Y_1 2$，$Y_2 2$，$Y_3 2$）	$\overline{\Delta s_1} = 1.49831\times10^{-4}$, $\overline{\Delta s_2} = 1.87129\times10^{-5}$	$\boxed{\overline{\Delta s_1} = 1.98616\times10^{-2}}$, $\overline{\Delta s_2} = 1.87129\times10^{-5}$
（$S3$，$Y_1 3$，$Y_2 3$，$Y_3 3$）	$\overline{\Delta s_1} = 1.69067\times10^{-9}$, $\overline{\Delta s_2} = 5.96740\times10^{-8}$	$\overline{\Delta s_1} = 1.03223\times10^{-4}$, $\overline{\Delta s_2} = 5.96740\times10^{-8}$
（$S4$，$Y_1 4$，$Y_2 4$，$Y_3 4$）	$\overline{\Delta s_1} = 1.01271\times10^{-6}$, $\overline{\Delta s_2} = 5.43897\times10^{-8}$	$\overline{\Delta s_1} = 1.01261\times10^{-6}$, $\overline{\Delta s_2} = 5.43897\times10^{-8}$
（$S5$，$Y_1 5$，$Y_2 5$，$Y_3 5$）	$\overline{\Delta s_1} = 8.58888\times10^{-7}$, $\overline{\Delta s_2} = 1.43737\times10^{-7}$	$\boxed{\overline{\Delta s_1} = 6.10469\times10^{-2}}$, $\overline{\Delta s_2} = 1.43737\times10^{-7}$
（$S6$，$Y_1 6$，$Y_2 6$，$Y_3 6$）	$\overline{\Delta s_1} = 1.46789\times10^{-9}$, $\overline{\Delta s_2} = 7.44673\times10^{-8}$	$\overline{\Delta s_1} = 8.64704\times10^{-5}$, $\overline{\Delta s_2} = 7.44673\times10^{-8}$
（$S7$，$Y_1 7$，$Y_2 7$，$Y_3 7$）	$\overline{\Delta s_1} = 2.09947\times10^{-6}$, $\overline{\Delta s_2} = 5.09306\times10^{-8}$	$\boxed{\overline{\Delta s_1} = 5.47477\times10^{-1}}$, $\overline{\Delta s_2} = 5.09306\times10^{-8}$
（$S8$，$Y_1 8$，$Y_2 8$，$Y_3 8$）	$\overline{\Delta s_1} = 1.70404\times10^{-4}$, $\overline{\Delta s_2} = 2.87375\times10^{-7}$	$\boxed{\overline{\Delta s_1} = 1.42761\times10^{-2}}$, $\overline{\Delta s_2} = 2.87375\times10^{-7}$
（$S9$，$Y_1 9$，$Y_2 9$，$Y_3 9$）	$\overline{\Delta s_1} = 5.72487\times10^{-7}$, $\overline{\Delta s_2} = 2.03018\times10^{-7}$	$\boxed{\overline{\Delta s_1} = 1.62891\times10^{-2}}$, $\overline{\Delta s_2} = 2.03018\times10^{-7}$
（$S10$，$Y_1 10$，$Y_2 10$，$Y_3 10$）	$\overline{\Delta s_1} = 1.88750\times10^{-4}$, $\overline{\Delta s_2} = 1.60825\times10^{-5}$	$\overline{\Delta s_1} = 3.42968\times10^{-4}$, $\overline{\Delta s_2} = 1.60825\times10^{-5}$

It should be noted that the defects of mutant1 were detected by Flag1. Though the perimeter of the triangle, Flag2 of metamorphic relationship is gained. Area of a triangle is counted by the mutant1's function. Therefore, the detection of mutant1 cannot be detected by taking advantage of Flag2.

5. Conclusions

This paper presents the method of integer bugs detection based on metamorphic relationship. It is testified by experiments that this method detects invisible abnormal failures that cannot be detected by traditional testing techniques. This paper has some pertinence to the initial test input and the selection of metamorphic relationship.

However, for a metamorphic relation, detecting several integer defects is simple, but it is hard to choose suitable metamorphic relations to detect whole integer defects. So testers need to acquire and choose suitable metamorphic relations in order to detect more integer defects.

Nowadays, some researchers combine evolutionary algorithm with metamorphic relationship to improve the probe of bugs efficiency of the test case set [10]. However, whether this way is valid for the integer bugs detection remains to be further researched.

References

1. Yi, Yao, Zheng Changyou, Huang Song, and Ren Zhengping. "Research on Metamorphic Testing: A Case Study in Integer Bugs Detection", 2013 Fourth International Conference on Intelligent Systems Design and Engineering Applications, 2013.
2. L. Baresi and M. Young, “Test oracles,” Tech. Rep. CIS-TR01-02, Department of Computer and Information Science, University of Oregon, Eugene, Ore, USA, 2001.
3. T.Y. Chen, S.C. Cheung and S.M. Yiu, Metamorphic testing: a new approach for generating next test cases, Technical Report HKUST-CS98-01. (1998).
4. Chen, H.Y., Tse, T.H., and Chen, T.Y. TACCLE: a methodology for object-oriented software testing at the class and cluster levels, ACM Transactions on Software Engineering and Methodology, (2001), Vol. 10, pp.56–109.
5. Kuo, F.C., Liu, S., and Chen, T.Y. Testing a binary space partitioning algorithm with metamorphic testing. SAC '11 Proceedings of the 2011 ACM Symposium on Applied Computing ACM New York, NY, USA, (2011), pp.1482-1489.
6. hen, T.Y., Ho. J. W.K., Liu H., and Xie, X. An innovative approach for testing bioinformatics programs using metamorphic testing. BMC Bioinformatics. (2009), Vol. 10, pp.24-35.
7. Kuo, F.C., Chen, T.Y., Tam, .W.K. Testing embedded software by metamorphic testing: A wireless metering system case study. 2011 IEEE 36th Conference on Local Computer Networks. Bonn, (2011), pp.291-294.
8. Asrafi, M., Liu, H., and Kuo, F.C. On Testing Effectiveness of Metamorphic Relations: A Case Study. 2011 Fifth International Conference on Secure Software Integration and Reliability Improvement, IEEE Computer Society Washington, DC, USA, (2011), pp.147-156.
9. P. E. Ammann and J. C. Knight, “Data diversity: an approach to software fault tolerance,” IEEE Transactions on Computers, vol. 37, no. 4, pp. 418–425, 1988.
10. G. W. Dong, S. Z. Wu, G. S. Wang, T. Guo, Y. G. Huang, Security Assurance with Metamorphic Testing and Genetic Algorithm. IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, (2010), pp.368-373.