Mutation Testing:An Oversight of Equivalent Mutant Problem

Rupinder Kaur
Research Scholar, Department of Computer Science & Applications, Kurukshetra University, Kurukshetra, Haryana
Email: kaur.rupinder036@gmail.com
Sanjay Tyagi
Assistant Professor, Department of Computer Science & Applications, Kurukshetra University, Kurukshetra, Haryana
Email: tyagikuk@yahoo.com

Abstract
Mutation testing is an approach which will help to find software errors. Mutation testing creates different versions of a program by making small syntactic changes,known as mutants of program.However, the mutants functionally identical tomainversion,are well-known as equivalent mutants.The topic of equivalent mutants is one of major problem of mutation testing as they remain undetected in program by any test suite. Traditionally, equivalent mutants were detectedmanually,thus making testing more time consuming and difficult. Nowadays, various algorithms,mutation operators and tools are implemented to achieve a solution to the equivalence problem to some extent.Butautomatic detection of alltheequivalent mutant is still a problem.This paper gives an overview of various types of equivalent mutants,methods to overcome this problem and oversight of tools and algorithms to detect such mutants.
Keywords-equivalent mutant, mutation, mutation operator, mutation testing,fault detecting technique.

1. Introduction

Mutation testingisan error revealing approach for software testing[1][2]. Mutation testing induces errors into the program and generates mutant of the program. The technique thus givesa test case adequacy criteria and helps in the revelation of errors in programs. Amutant is believed to be dead, when a test input finds difference between erroneous and the original program.If no test suite can reveal the variation between original and mutant program,then mutant is still live. Live mutants can be of two types, one which can be killed by improving test data and another which are identical to original program.Live mutants,functionallyequal to original program, are called equivalent mutants.These equivalent mutants remain undetected during testing and is a major problem in mutation testing.The equivalence detection is an undecideable process. Mutation testing measures effectiveness of a test data by providing a mutation score(MS).

$$MS = \frac{MD}{MC - ME}$$

MC= total mutants created,
MD = deadmutants,
ME = number of equivalent mutants.

Undetected identical mutants,will never lead to 100%mutation score.Thus,the tester will not have full faith on test data and remains wondered if the live mutants are equal or the test suites are inadequate to expose errors. Detection of equivalent mutants manually is quite expensive and time consuming and thus making mutation testing expensive too.
Fig.1 shows the program and its mutant. The statement 4 in original program is changed to formmutantm. Mutant m and original program is equivalent as mi and ghave same value,resulting in same output.

Program	mutant (m)
int fun fmin(g,h)	int fun fmin(g,h)
intg,h	intg,h
mi=g	mi=g
if(h<g)	if(h<mi)
mi=h	mi=h
return(mi)	return(mi)

Figure 1.Example ofequivalent mutant

2. Types of equivalent mutants

There are threekinds of equivalent mutants[3]:
1. Expression equivalent mutant
2. Pre-condition equivalent mutant
3. Weak equivalent mutant

2.1Expression equivalent mutant

In Fig.2,the statement 5 is changed in original program to formmutantm1.It is easy to examine the equivalence in m1 and original program because the program's state after or beforethe execution of mutated statement is need not to be taken into account.The comparison of value of M in mutant and original program is only need to consider to findequivalence in this case. These kindsof equivalent mutants are expression equivalent mutants.

Original program	mutant(m1)
Real fun G(L,M)	Real fun G(L,M)
Real(L,P)	Real(L,P)
Integer M,J	Integer M,J
P=L	P=L
IF(M.LT.1) THEN	IF(M.LT.0.5) THEN
M=1	M=1
ENDIF	ENDIF
RETURN(M)	RETURN(M)

Figure 2.Expression equivalent mutant

2.2Precondition equivalent mutant

In Fig.3, m2is mutant of original program.The detection of equivalenceof m2with the original program needs to observe the state of program beforetheexecution of mutated statement. The precondition need to be satisfied is that C cannot be less than 1. Hence, the mutantsareprecondition equivalent mutant.

2.3 Weak equivalent mutant

An another kind of mutant is m3, shown in Fig.4. Negative value of L leads to difference in P's value in mutant from original program which causes difference in state in the original and mutant program. Investigation of equality for this kind of mutant requires running the program beyond the fault statement and finding whether the variation in state goes to exit point. These kinds of mutants are weak equivalent mutants.

Original program	mutant(m2)
Real funG(A,C)	Real fun G(A,C)
Real A,B	Real A,Y
Integer C,I	Integer C,I
B=A	B=A
DO LB J=1,C	DO LB J=1,ABS(C)
B=B*B	B=B*B
LB IF(B.GT.0.5)	LB IF(B.GT.0.5)
Q=TRUE	Q=TRUE
ELSE	ELSE
Q=FALSE	Q=FALSE
END	END

Figure 3.Precondition equivalent mutant

Original program	mutant(m3)
Real fun G(L,M)	Real fun G(L,M)
Real L,P	Real L,P
Integer M,J	Integer M,J
P=L	P=NEG(L)
DO L1 J=1,M	DO L1J=1,M
P=P*P	P=P*P
IF(P.GE.1)	IF(P.GE.1)
Q=L+P	Q=L+P
ELSE	ELSE
Q=L-P	Q=L-P
END	END

Figure 4.Weak equivalent mutant

3.Approaches to overcome Equivalent Mutant Problem (EMP)

There are three broad categories in which approaches to overcome EMP are distributed[4].

1. Detecting equivalent mutant techniques
2. Avoiding equivalent mutant generation techniques
3. Suggesting equivalent mutants techniques

3.1 Detecting equivalent mutant techniques

Following techniques can only detect equivalent mutants:

- Compiler optimization techniques [5]
- Mathematical rules [6]
- Program slicing [7]
- Margrave's change-impact analysis [8]
- Lesarmodel-checker [9]

3.2 Avoiding equivalent mutant generation techniques

Following are the techniques to avoid generation of equivalent mutant.

- Selective mutation [3]
- Program dependence and mutation analysis relationship [10]
- Co-evolutionarytechniques with selective mutation [11]
- Isolation of 1st order equivalent mutants using 2nd order mutation [12]
- Higher order mutation testing [13]

3.3 Suggesting equivalent mutants techniques

Followingare the suggestion techniques for equivalent mutants:

- Equivalent mutant's impact on coverage [14]
- Changes in coverage to recognize equivalent mutants [15]

4.Literature Review

Previous approaches for automatically detecting equivalent mutants were discovered and new algorithms had been presented to automatically reveal equivalent mutants under specified conditions[5]. Some of equivalent mutants could be foundby using data flow and compiler optimizationtechniques.The algorithm designed had been used to build a tool called Equalizer and embed this tool into Mothra testing system to detect equivalent mutants.Although all equivalent mutants were not possible to detect but still Equalizer was able to detect equivalent mutants to some extent in several programs,almost half in some cases.As this problem was handled manually before,so this tool is a partial solution to equivalent mutant problem. The results from this paper can be useful to tester and helps to make mutation testing more useful on practical basis.

A technique using mathematical constraints[6]was presented to identify equivalent mutants and infeasible paths automatically. Specific algorithms proposed became a good partial solution to equivalent mutant problem.The results were even better when the technique was exercised to the feasible path problem. Equivalencer,a tool was able to find approximately 45%of equivalent mutants.Apowerful automated test environment was provided to produce highly assured software at affordable cost.The system would allow a user to provide some input and helped to find errors on basis of input-output pairs.

Program slicing [7]reduced the efforts in determining equivalent mutants as most of time and cost is associated with manual handling of mutants that are equal or hard to reveal error.Program simplification process simplified the program where the mutant was not equivalent andhelped to kill that mutant. To reduce equivalent mutants, Program slicing could also be used with firm and strong mutation.The mutants were sliced for correlatingthe influence of main and mutant code on specificoperands.Amorphous slicing was used and compared with conventional slicing.Smaller slices are produced by amorphous slicing than conventional slicing.

A fitness function[8]was designed to detect equivalent mutants.Also it was shown,how to choose effective test cases and mutants of original program.The method proposed did not refuse selective mutation or reduced number of mutants.Ingiventechnique,afterapplying all mutation operators,a pool of mutants was produced from original program.A GA evolved subsets of mutant and insignificant or low performance mutants were rejected.Similarly, a GA was suggested for test data to increase testing capabilitywith adequacy score.At the end, the two methods were integratedfor evolvement of mutants and test data parallely.

Previous work considered cost benefits of selective mutation whereas [3] considered detecting equivalent mutant's cost. The cost was measured by considering equivalent mutants and statements in program. Finding most efficient operator,operators were compared by using score and cost.The comparison of score and cost of efficient operators were made with another set of operators and selective mutation.x%selective mutation was more efficient than selective mutation depending uponexpression and {abs, aor, san, sdl, uoi } group of operators, when a

mutation score near to 100% was needed. Also, selective mutation depending upon strictgroup of effective operators(aor, san, sdl, ror, uoi) was efficient when less stringent test coverage was required.

To find equivalent and partially equivalent mutants, nine problematic patterns were introduced[9]. Specific conditions between definitions and uses of variables were introduced for each pattern. For data flow analysis, single static assignment was utilized. 70% of equivalent mutants were revealed by using this technique.

I-EQM method was able to isolate equivalent mutants [10]. Given first order mutants were classified as alive or killed by utilizing second order mutants. The obtained results revealed that 82% of killable mutants were classified correctly with 71% precision.

Mutation impact coverage was used to separate equivalent mutant from non-equivalent mutants. The implementation and deployment of technique was easy. Schuler *etal.*claimed that if a mutation changes coverage, 75% chance of its being non-equivalent[11].

5.Conclusion

This paper discussesmutation testing and the problems in mutation testing.Main emphasis is on equivalent mutant,which is a major problem in mutation testing.Various types of equivalent mutants are also discussed.Methods to overcome equivalent mutants are described in order to detect and avoid equivalent mutants.Algorithms and tools have also been discussed to overcome the equivalent mutant problem.There is still a lot of work to be done in this field.Various partial solutions were developed. Still work is going on to find a complete solution to equivalent mutant problem.The solutions given in literature review will help in future to make mutation testing more widely use in practice.

References

[1] R. A. Demillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection:Help for the practicing programmer," *IEEE Computer*, pp. 34-41, 1978.

[2] R. G. Hamlet, "Testing programs with the aid of a compiler," *IEEE Transactions on Software Engineering*, pp. 279-290, 1977.

[3] Elfurjani S. Mresa and Leonardo Bottaci, "Efficiency of mutation operators and selective mutation strategies:An empirical study," *Software Testing,Verification and Reliability*, pp. 205-232, December,1999.

[4] Lech Madeyski, Wojciech Orzeszyna, Richard Torkar, and Mariusz Jozala, "Overcoming the Equivalent Mutant Problem:A Systematic Literature Review and a Comparative Experiment of Second Order Mutation," *IEEE TRANSACTIONS on SOFTWARE ENGINEERING*, 2013.

[5] A. Jefferson Offut and W. Michael Craft, "Using Compiler Optimaztion Techniques to Detect Equivalent Mutants," *Software Testing,Verification and Reliability*, pp. 131-154, 1994.

[6] A. Jefferson Offut and JIE PAN, "Automatically Detecting Equivalent Mutants and Infeasible Paths," *Software Testing,Verification and Realiability*, pp. 165-192, 1997.

[7] Rob Hierons, Mark Harman, and Sebastian Danicic, "Using Program Slicing to Assist in the Detection of Equivalent Mutants," *Software Testing,Verification and Reliability*, pp. 233-262, 1999.

[8] Evan Martin and Tao Xie, "A Fault Model and MutationTtesting of Access Control Policies," in *16th International Conference World Wide Web*, 2007.

[9] Lydie du Bousquet and Michel Delaunay, "Mutation analysis for Lustre programs," *IEEE*, 2007.

[10] Mark Harman, Rob Hierons, and Sebastian Danicic, "The Relationship between Program Dependence and Mutation Analysis," in *Mutation testing for the new century*.: Kluwer Academic Publishers.

[11] Konstantinos Adamopoulos, Mark Harman, and Robert M. Hierons, "How to Overcome the Equivalent Mutant Problem and Achieve Tailored Selective Mutation Using Co-evolution," in *GECCO*, Heidelberg, 2004, pp. 1338-1349.

[12] Marinos Kintis, Mike Papadakis, and Nicos Malevris, "Isolating first order equivalent mutants via second order mutation," in *IEEE Fifth International Conference on Software Testing,Verification and Validation*, 2012, pp. 701-710.

[13] Yue Jia and Mark Harman, "High Order Mutation Testing,".

[14] Bernhard J.M. Grun, David Schuler, and Andreas Zeller, "The impact of equivalent mutants," in *IEEE International Conference on Software Testing Verification and Validation Workshops*, 2009.

[15] David Schuler and Andreas Zeller, "(Un-)Covering Equivalent Mutants," in *IEEE Third international conference on software testing,verification,and validation*, washington,USA, 2010, pp. 45-54.

[16] Marinos Kintis and Nicos Malevris, "using data flow patterns for equivalent mutant detection," in *IEEE International Conference on Software Testing,Verification,Validation* , 2014.