A Review On: Code Transformation Techniques for Decision Coverage

Abhishek Sahu[1], Rahul Kumar Chawda[2]
[1] Student of MCA, Kalinga University, Raipur
[2] Assistant Professor, Department of Computer Science, Kalinga University, Raipur

Abstract :- Modified Condition / Decision Coverage (MC / DC) is a white box testing criteria aiming to prove that all conditions involved in a predicate can influence the predicate value in the desired way. In regulated domains such as aerospace and safety critical domains, software quality assurance is subjected to strict regulations such as the DO-178B standard. Though MC/DC is a standard coverage criterion, existing automated test data generation approaches like CONCOLIC testing do not support MC/DC. To address this issue we present an automated approach to generate test data that helps to achieve an increase in MC/DC coverage of a program under test. We use code transformation techniques for transforming program. This transformed program is inserted into the CREST TOOL. It drives CREST TOOL to generate test suite and increase the MC/DC coverage. Our technique helps to achieve a significant increase in MC/DC coverage as compared to traditional CONCOLIC testing's.

Keyword- CONCOLIC testing, Code transformation techniques, MC/DC

Introduction - Software engineering proposes systematic and cost-effective methods to software development process . These methods have resulted from innovations as well as lessons learnt from past mistakes. Software engineering as the engineering approach to develop software. Software is usually subject to several types and cycles of verification and test. In the early days of software development, software testing was considered only a debugging process for removing errors after the development of software.

Software Testing

Software Testing is a process that detects important bugs with the objectives of having better quality software. This is the way to increase reliability of software projects [2]. The technique software testing is responsible for achieving good quality software and high software dependability. Software testing consists of the steps of execution of a system under some conditions and compares with expected results [3]. The conditions should have both normal and abnormal conditions to determine any failure under unexpected conditions.

Software Testing Goals

The main goals of software testing are divided into three categories and several subcategories as follows:

1. Immediate Goal :

ˆ Bug Discovery,

ˆ Bug Prevention

2. Long-term Goals:

ˆ Reliability,

ˆ quality,

ˆ customer,

ˆ satisfaction,

ˆ risk management

3. Post Implementation Goals:

ˆ Reduced maintenance cost,

ˆ Improved testing process

Software Testing Life Cycle

The testing process divided into a well-defined sequence of steps is termed as a software testing life cycle (STLC). The STLC consists the following phases:

- Test planning,
- Test design,
- Test execution
- Test review/post execution.

Software Testing Techniques

In software world it has been noticed that 100% efficient software testing is not possible. But an effective testing can solve this problem but to follow the effective testing is very difficult. The method to determine effective test case is known as Software Testing Technique. Two objectives are making the effective test cases that are detection of numbers of bugs and coverage of testing area. The different levels of testing Unit testing, Integration testing, Function Testing, System testing, and Acceptance testing. The detailed testing stages are followed:

1. Unit Testing: Each System Component of whole software is individually tested for all functionality and its interfaces.
2. Integration Testing: Process of mixing and testing multiple building blocks together. The individual tested component, when mixed with other components, is untested for interfaces. Therefore it may have bugs in integrated workspace. So, the purpose of this testing is to uncover this bug.
3. Function Testing: To measure systems functional component quality is the main purpose of functional testing. This is to expand the bugs related to problems between system behavior and specifications.
4. System Testing: Its objective is not to test particular function, but it tests the system on various platforms where bugs exist.
5. Acceptance Testing: This technique used by customer after software developed. Compares the process of the final status of project and agreement of acceptance criteria performed by the customer.

Software Testing Strategies

Testing strategies are mainly divided into two categories:

1. Black Box Testing: The structure of software is not considered only the functional requirements of the module are taken under consideration. In this the software system act as a black box taking input test data and and giving output results.
2. White Box Testing: As everything is transparent in glass like that in this software it visible in all aspects it is called as glass box testing. Structure, design and code of software should be studied for this type of testing. Also it is called as development or structural testing

Methodology

Below, we discuss some relevent definitions that will be used in our approach.

1. Condition: Boolean statement without any Boolean operator is called as condition or clause.
2. Decision: Boolean statement consisting of conditions and zero or many Boolean operators is called as decision or predicate. A decision with no Boolean operator is a condition . Example: Let's take an example: if((a>100) && ((b<50) ||(c>40)) Here, in the if-statement whole expression is called as predicate or decision, && and ||are the Boolean operators and (a > 100), (b < 50) and (c > 40) are different conditions or clause.
3. Group of Conditions: Boolean statement consisting of two or more conditions and one or more operators is called as a group of conditions. Example: statement1: if ((A && B) ||(C && D)). Here A, B, C, D are four different conditions and (A && B), (C && D) are two groups of conditions. Statement 1 is nothing but the decision statement.
4. Logic Gates: They are the fundamental building blocks of digital electronics and perform some logical functions. Most of the logic gates accept two binary inputs and result in single output in the form of 0 or 1. show the truth table for two and three variables respectively .

Modified Condition/ Decision Coverage

MC/DC was designed to take the advantages of Multiple Condition testing when retaining the linear growth of the test cases. The main purpose of this testing is that in the application code each and every condition in a decision statement affects the outcome of the statement . MC/DC needs to satisfy the followings:

- ˆ Each exit and entry point in the code is invoked.
- ˆ Each and every condition in a decision statement is exercised for each possible output.
- ˆ Each and every possible output of every decision statement is exercised.
- ˆ Each and every condition in a statement is shown to independently affect the output of the decision stated.

Following five steps are used to determine the MC/DC coverage:

1. Develop a proper representation of the program.
2. Find the test inputs, which can be obtained from the requirement based tests of the software product.
3. Remove the masked test cases. The masked test case is one whose output for a particular gate hidden from all others outputs.
4. Calculate MC/DC .
5. At last the results of the tests are used to confirm correct operation of the program. For the details of constructing the MC/DC table the readers may refer to.

Conclusion

We have proposed a novel approach to automatically increase the MC/DC coverage of a program under test. Here we have presented an approach to automate the test data generation procedure to achieve increased MC/DC coverage. We have used existing CONCOLIC tester i.e crest tool with a code transformer based on sum of product (SOP) boolean logical concept to generate test data for MC/DC. In the following, we summarize the important contributions of our work.

References

1. Z. Awedikian, K. Ayari, and G. Antoniol, "Mc/dc automatic test input data generation," in Proceedings of the 11th Annual conference on Genetic and evolutionary computation, GECCO '09, (New York, NY, USA), pp. 1657–1664, ACM, 2009.
2. K. Hayhurst, D. S. Veerhusen, J. J. Chilenski, and L. K. Rierson, "A practical tutorial on modified condition/decision coverage," 2001.
3. S. B. Akers, "On a theory of boolean functions," pp. 487 – 498, Journal Society Industrial Applied Mathematics, 7(4), December 1959.
4. A. L. White, Programming Boolean expressions for testability, pp. 3110–3122. IEEE, 2004.
5. K. Sen, D. Marinov, and G. Agha, "Cute: a concolic unit testing engine for c," in In ESEC/FSE-13: Proceedings of the 10th European, pp. 263–272, ACM, 2005.
6. J. C. King, "Symbolic execution and program testing," Commun. ACM, vol. 19, pp. 385–394, July 1976.
7. M. Kim, Y. Kim, and Y. Choi, "Concolic testing of the multi-sector read operation for flash storage platform software," Under Consideration for publication in Formal Aspects of Computing, 2011. CS Dept. KAIST, Daejeon,South Korea and School of EECS, Kyungpook National University, Daegu, South Korea.
8. J. J. Chilenski and S. P. Miller, "Applicability of modified condition/decision coverage to software testing," Software Engineering Journal, vol. 9, no. 5, pp. 193–200, 1994.
9. P. McMinn, "Search-based software test data generation: a survey: Research articles," Softw. Test. Verif. Reliab., vol. 14, pp. 105–156, June 2004.
10. C. Cadar, V. Ganesh, P. M. Pawlowski, D. L. Dill, and D. R. Engler, "Exe: automatically generating inputs of death," in Proceedings of the 13th ACM conference on Computer and communications security, CCS '06, (New York, NY, USA), pp. 322–335, ACM, 2006.
11. D. L. Bird and C. U. Munoz, "Automatic generation of random self-checking test cases," IBM Syst. J., vol. 22, pp. 229–245, Sept. 1983.
12. C. Pacheco, S. K. Lahiri, M. D. Ernst, and T. Ball, "Feedback-directed random test generation," in ICSE '07: Proceedings of the 29th International Conference on Software Engineering, (Minneapolis, MN, USA), IEEE Computer Society, 2007.