International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-3 Number-3 Issue-11 September-2013

Measuring Coverage Percentage for C Programs using Code Slicer and CREST Tool

Sangharatna Godboley[1], Avijit Das[2], Kuleshwar Sahu[3] , Durga Prasad Mohapatra[4], Banshidhar Majhi[5]

Abstract

Augmented test suite generation is a technique to minimise test effort and duration. Modified condition and decision coverage (MC/DC) is a white box software testing criteria targeting to prove all the conditions involved in a predicate which can influence the predicate value in an efficient way. The coverage analysis is a structural testing method, which helps to remove gaps in a test suite and determines when to stop testing. In this paper, we propose an augmented method to generate a test suite that helps in measuring coverage percentage of a program. We propose a technique which consists of mainly three modules. The crest module is a program slicer, who accepts a program written in C language and uses some slicing criteria results an executable sliced program. The second module is the CREST tool (CONCOLIC tester) which accepts the executable C sliced program as an input. The CREST tool drives to generate the test suite. The third module is Coverage Analyser (CA) to compute the coverage percentage. Our technique helps to achieve the coverage percentage with time taken to execute the program.

Keywords

Code Slicer, Crest Tool, Concolic Testing, Coverage Analyser, Program Slicing, And Modified Condition / Decision Coverage.

1. Introduction

Testing strategies are mainly divided into two categories, BLACK BOX TESTING: The structure of software is not considered only the functional requirements[1] of the module are taking under consideration. The software system act as a black box, taking input test data and giving output results. WHITEBOX TESTING: As everything is transparent in glass, like that visibility in all aspect for software shows the property of glass box testing. Structure, design and code of software [7] should be studied for this type of testing. Also it is called as development or structural testing. Modified condition / decision coverage follows four criteria: The condition coverage and decision coverage criteria by resulting that each condition [8] in a decision independently affects the output of the decision, each decision has to exercise for all results at least once in whole performance, each condition or clause in the decision has to exercise all possible results at least once in whole performance and each clause in the predicate has to independently affect the predicate's results. The main objective of our work is to develop an automated approach to generate [4] test suite that can evaluate MC/DC coverage percentage. To reach our objective, we propose the approach to calculate coverage percentage after using CONCOLIC tester CREST tool. The CONCOLIC testing is combination of concrete and symbolic [5] testing was originally designed to achieve branch coverage. In our work, we present the code slicer as a crest module in which we insert program code under test written in C language and obtained the sliced program as output. Sliced program is a subset of original program. For slicing the program we need to provide slice criteria. Code slicer consists of four steps, crest step is converting code into an abstract syntax tree (AST), second step is converting the AST into a RCFG, third step is implementing a slicing algorithm using the RCFG, and fourth step is converting the slicing algorithm output back into the original program’s syntax. The second module of approach is generating test suite using CONCOLIC tester. The CONCOLIC tester generates concrete input values for the sliced code. The CREST tool is open source CONCOLIC tester to generate test suite for C language program. Third module is coverage analyser, it calculates the coverage percentage. We need to provide MC/DC test data for each and every clause with original program to coverage analyser and at last we get the coverage percentage. In our observations when we

[1]**Sangharatna Godboley**, Department of Computer Engineering, ARMIET Shahpur, Thane, India.
[2]**Avijit Das**, Defence Research & Development Organisation, Hydrabad India.
[3]**Kuleshwar sahu**, Department Computer Science, National Institute of Technology Kurukshetra, India.
[4]**Durga Prasad Mohapatra**, Department Computer Science and Engineering, National Institute of Technology Rourkela, India.
[5]**Banshidhar Majhi**, Department Computer Science and Engineering, National Institute of Technology Rourkela.

are inserting sliced program to CONCOLIC tester some MC/DC test data are generate, using these values and our program we calculates the coverage percentage. The second section of our paper will discuss about some basic concepts. The third section shows the related works. The fourth section discusses about our proposed approach to measure coverage percentage. The fifth section shows the implementation for our concept. The sixth section concludes our approach. At last the reference for our paper.

2. Basic Concepts

In this section, we discuss some basic definitions regarding program slicing, MC/DC coverage, and CONCOLIC testing.

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

a. Develop a proper representation of the program.
b. Find the test inputs, which can be obtained from the requirement based tests of the software product.
c. Remove the masked test cases. The masked [6] test case is one whose output for a particular gate are hidden from all others outputs.
d. Calculate MC/DC.
e. At last the results of the tests are used to confirm correct operation of the program.

2.Program Slicing: Program slicing [9] is a decomposition method, which searches the portion of a program that have semantic importance to a chosen point of interest, called as the slice criterion, extracting these parts to form the program slice, which is a subset of original program. Program slicing reduces code to statements relevant for partial computation, deleting irrelevant statements. Sliced program is a subset of original program. A program slice consists of all the statements affecting the variable(s) at a position in the program, which are both specified by the slice criteria. Program slicing is divided in two main categories: semantic and syntactic. The semantic types include the static slicing, dynamic slicing, and conditioned slicing while the syntactic types include syntax preserving and amorphous slicing. These two elements are important aspects of slicing. The semantic element describes that, what portion of the original program is to be preserved. The syntactic element consists of two possible types of program slicing. The crest type is, the program's original syntax can be preserved, removing sections of the program that have no effect in the semantics of interest, and the second type is the syntax transformation take place, which preserve the semantic detail of the program.

3. CONCOLIC Testing: The CONCOLIC testing [3] combines the concrete constraints and symbolic constraints to automatically generate test data for full path coverage. CONCOLIC testing produces test suites by executing the program with random values. The CONCOLIC testing producing test cases by executing the program code with random value. The CONCOLIC tester selects a value from the path constraints and negates the values to create a new path value. Then the CONCOLIC tester finds concrete constraints to satisfy the new path values. These values are inputs for all next execution. This process performed iteratively until exceeds the threshold value or sufficient code coverage obtained.

3. Related Works

Bokil et al. [2] proposed a tool *AutoGen* that reduces the cost and effort for test data preparation by automatically generating test data for C code. AutoGen takes the C code and a criterion such as statement coverage, decision coverage, or Modified Condition / Decision Coverage (MC/DC) as input and generates non-redundant test data that satisfies the specified criterion.

Samer Hamood [12] has designed C Slicer. Hamood proposed a parallel slicing algorithm, which was adopted to compute slices, and inadvertently lead to the innovation of a new sequential algorithm based upon the old parallel one.

Awedikian et al. [15] have given a concept for automatic MC/DC test generation. They used ET methods to generate test inputs to achieve MC/DC coverage. Their objective was MC/DC coverage. However, a drawback of local maxima as the HC algorithm performs data search in limited scope. This shows the solution is not globally optimal.

Chen et al. [14] have written a paper to compute test Coverage, the approach is a gradation model, in which different coverage has different ranks, and the test coverage [9] of the upper layer is computed according to the coverage of all the layers from the lowest to current layer and the rank difference. To compare the importance of different variables, the paper proposes new concept coverage about variables, based on program slicing, and adds powers according to their importance. They have focus on the important variables to obtain higher test coverage. In most cases, the coverage obtained by our method is bigger than that obtained by a traditional concept, because the coverage about a variable takes only the codes related into account, and the gradation model takes more factors into consideration when analyzing test coverage.

4. Measuring of coverage Percentage

This section explains an explanation of the proposed approach. Evaluation of coverage percentage metric using code slicer and CREST tool [16,17]. Basically our approach based on three modules: CODE SLICER, CONCOLIC TESTER [16,17], and COVERAGE ANALYSER[16,17]. Figure 1 shows the concept of our approach by combination of all three modules:

1. CODE SLICER: Code slicer accept an error less C written program code to be slice. The C program passed to slicing mechanism. The slicer mechanism is responsible for transforming the original C program into the sliced program. There is a slicing criteria need to provide to slice a C program. We have to specify a slice criterion by variable(s) and statement through inputting those values. The program will then be redisplayed after the slicing process complete. The program code will be in sliced version of the original program code. From Figure 2, the C program to be sliced is the input of the approach that is entered. The code is then read into the parser. The code is parsed and transformed into its AST. The AST is passed to the slicer mechanism, starting the slicing approach by supplying the code slicer with all the necessary information on the code it will slice. At last, the code slicer results the sliced C program in the form of a C syntax.

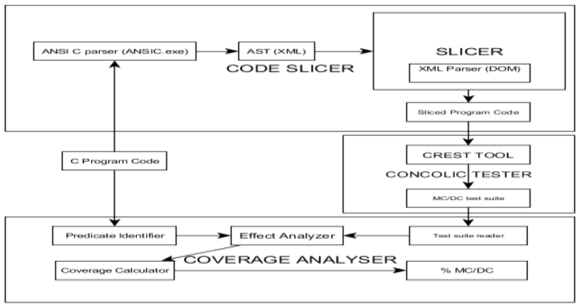

Figure 1: Schematic Representation of CS, CT, AND CA

The main element of CODE SLICER is SLICER (XML Parser (DOM)). The SLICER functionality is depends on three categories: CODE CONVERTER, CODE EXTRACTOR, and SLICER TOOL. The Code Converter is crest category. The Code Converter category is responsible for converting the original C program code into a CFG graph representation.

This process is happened by running the parser. The parser captures the AST XML output in an XML file. The code converter creates if the C program code is errorless otherwise exception is thrown. The converted XML _le is then passed to the XML parser DOM is so that the AST XML can be traversed. When AST is traversing the program statements are rebuilt. Control Flow Node class model CFG nodes, to which Code Converter passes a Syntax class. They are being constructed while traversing the AST. The Code Converter calls a method from Code Extractor to calculate the token points which are stored in the syntax class. When all Node objects have assembled, the code converter is able to compute a reverse control owe graph *(rcfg)* technique as a Control Flow Graph data structure object. The second category of slicer is code extractor. It is architecturally straighter than the code converter. The code extractor is dependent on just one other module, and basically writes and reads to less. The code extractor points the exact location in the program files of every statement token. Then the statement is passed, and begins and end indices to Token Point data structure object. Token Point object array is returned. The line statement is on attained from the AST, and together with the statement program code. The statement line provide a starting point to locates they began and end indices of every tokens. The code extractor's another function performing the creation of file that saves the slice syntax. The placed values stored in Token Point objects to allow code extractor to copy the same syntax found in the input C program into the new slice file. The Slice Tool is third category of SLICER. It is the combination of two elements CODE CONVERTER and CODE EXTRACTOR. The main slicer's functionality of this SLICER TOOL is simple command line user interface. The C program code is to be sliced and slice criteria, and displaying on the command prompt screen the slicer's results as output. Here, the second category functionality CODE EXTRACTOR comes into play. Passing all these points to the write code and method that creates the file containing the slice, which Code Slicer Tool then displays for the users viewing.

2. CONCOLIC TESTER: The sliced version of original C program code generated from CODESLICER is passed to CONCOLIC tester CRESTTOOL. The CREST tool achieves branch coverage through any strategy for test suite generation. The CONCOLIC Tester is a combination of CONCrete and symbOLIC testing. The additional generated statements lead to generation of extra test suite for the sliced version of the program. There are several strategies like: DFS, CFG, and RANDOM etc. Due to random strategy, different execution of the CREST tool may not generate identical test data. The Test Case generation depends on the path of each and every execution of program. All test cases stored in .text files forms as a test suite. The input.txt files contains the selected input concrete values, this depends on strategy of selection values. The constraints solver selects the values. The CREST tool having constraints solver named as *yices*. The generation of numbers of input.txt file depends on both the iteration number provided and till all branches covered. The CREST tool record the covered node numbers in one .txt file named as coverage.txt file. From Figure 1 the second module represents the CONCOLIC TESTER. The CREST tool is the CONCOLIC tester which accepts sliced version of original C program code and generates MC/DC test suite as shown in Figure 1.

3. COVERAGE ANALYSER: From Figure 1, the third module represents Coverage Analyser which calculates the MC/DC coverage percentage achieved by a test suite. It is to evaluate the extent to which a C program code feature has been processed by test data. Also Coverage analyser ends inadequacy of test data and results an insight on those aspects of an implementation that have not been tested in whole execution. In our concept Coverage Analyzer is used to calculate coverage percentage metric performed by the test suite generated by the CONCOLIC tester CRESTTOOL and sliced program transformed by CODESLICER. The original C program code for testing and the test suite generated from CREST tool are passed to the Coverage Analyser. The coverage analyzer observe the extent to which independent effect of the component condition on the evaluating each predicate by the test data. The MC/DC percentage coverage achieved by the test cases for program as input by MC/DC coverage is calculated by the formula:

MC/DCcoverage=
((Total_independent_affected_conditions)/
(Total_condition_in_predicate))*100 (11)

5. Experimental Studies

We have proposed our approach using three module: CODE SLICER, CONCOLIC TESTER, and COVERAGE ANALYSER. The objective of CODE SLICER is to transform the original C program into sliced version. The CREST tool is a CONCOLIC TESTER to execute CONCOLIC testing for errorless C program code. We fetched the sliced program to CREST tool, then it gives test suite and coverage as an output. Then the generated test suite with original C program we calculate coverage percentage by COVERAGE ANALYZER. Table 1 and Table 2 shows the experimental results for original and sliced version of the C program.

Table 1: Experimental results I

Program	**Locs**	**Branches**	**Node**	**Branch Edges**	**Covered Branches**
PROGRAM1.C	16	10	17	12	7
PROGRAM1SLICED.C	12	2	7	0	2

Table 2: Experimental results II

Program	**Iterations**	**Test Data Files**	**Time (Sec)**	**Coverage Percentage**
PROGRAM1.C	8	8	3.00	75%
PROGRAM1SLICED.C	3	3	3.00	100%

From Table 1 and 2, we can see that the reduction in complexness of the sliced version program over the original program. The LOCs, EDGES, NODES and BRANCHES of the sliced version program are decreased. When we observe the time, we find that the time taken to execute both the programs is equal, but we achieve enhanced coverage percentage for the sliced version. From Table 2, we observed that, the percentage coverage for the original program without using code slicer is 75 %. But, the percent of coverage for the sliced program is 100 %, as we have used a code slicer. So, we may conclude that the percentage of coverage has been increased by 25% due to the use of code slicer. This is a significant increase in the percentage of coverage.

6. Conclusion and Future Work

In this work, we have proposed a novel approach to measure coverage percentage of a program written in

C language. Here we have presented an approach to automate [10] the test data generation [13] procedure to achieve coverage percentage with the time taken for execution. We have used the existing CONCLOICTESTER i.e CREST tool with a CODE SLICER to generate test data for MC/DC. Also, we have proposed an algorithm for coverage analysis which measures the coverage percentage. From the experimental results, we observed that, the sliced version program achieves 25%more coverage than the original version of program, due to the use of code slicer.

The future version of this paper will consist of code slicer with code transformer, to get the better result. We will have time analysis of execution.

Acknowledgment

We express our gratitude to Prof. Rajib Mall of IIT Kharagpur for providing the necessary inputs and guidance at different stages of our research work. We are very indebted to Mr. A. V. Gupta, Founder Trustee, and Dr. L. K. Bothra, Chairman, ARMIET Shahpur for their support.

```
Algorithm1:COVERAGE ANALYSER
Input:X,Test_Suite // Program X and Test_Suite obtained
Output:MC/DCcoverage // % MC/DC achieved for X
Begin
/*Identification of predicates*/
for each statement s∈X do
   if && or ||occurs in s then
      1 List_Predicate←adding_in_List(s)
   end if
end for
/* Determine the outcomes */
for each predicate p∈List_Predicate do
   for each condition c∈p do
      for each test_case t_d ∈ Test_Suite do
         if c evaluates to TRUE and calculate the outcome of p with t_d then
            2 True_Flag←TRUE
         end if
         if c evaluates to FALSE and calculate the outcome of p without t_d then
            3 False_Flag←TRUE
         end if
      end for
      if both True_Flag and False_Flag are TRUE then
         4 I_List←adding_in_List(c)
      end if
      5 C_List←adding_in_List(c)
   end for
end for
/* Calculating the MC/DC coverage percentage */
6 MC_DC_COVERAGE←(SIZEOF(I_List)/SIZEOF(C_List))× 100%
```

Figure 1: Coverage Analysis

References

[1] Qu, Xiao and Robinson, Brian. A case study of CONCOLIC testing tools and their limitations. 2011 International Symposium on Empirical Software Engineering and Measurement (ESEM), pages 117-126, Washington, D.C.,USA, 2011.
[2] Bokil, Prasad and Darke, Priyanka and Shrotri, Ulka and Venkatesh, R. Automatic test data generation for c programs. Third IEEE International Conference on Secure Software Integration and Reliability Improvement, SSIRI 2009. pages 359-368, 2009.
[3] Sen, Koushik and Marinov, Darko and Agha, Gul. CUTE: a CONCOLIC unit testing engine for C. In Proc. ESEC/FSE, ACM, 30(5), pages 263-272, Lisbon, Portugal, 2005.
[4] Burnim, Jacob and Sen, Koushik. Heuristics for scalable dynamic test generation. Proceedings of the 23rd IEEE/ACM international conference on automated software engineering, IEEE Computer Society, pages 443-446, Washington, D.C., USA, 2008.
[5] S. Krishnamoorthy, S M. Hsiao, and L. Lingappan. Strategies for scalable symbolic execution driven test generation for programs. In Science China Information Sciences, 54, pages 1797-1812, 2011.
[6] Suman, P. and Muske, T. and Bokil, P. and Shrotri, U. and Venkatesh, R.. Masking boundary value coverage: Effectiveness and efficiency Testing-Practice and Research Techniques,springer, pages 8-22, 2010.
[7] Kuhn, D Richard. Fault classes and error detection capability of specification based testing. ACM Transactions on Software Engineering and Methodology (TOSEM),ACM, 8(4), pages 411-424, 1999.
[8] Akers, S.B. On a theory of boolean functions. Journal Society Industrial Applied Mathematics, 7(4), pages 487-498, December 1959.
[9] Ammann,P. , Offutt, J. and Huang, H.. Coverage criteria for logical expression. In Proc. ISSRE, pages 99-107, Washington, D.C., USA, 2003.
[10] [Menno D. Hollander. Automatic Unit Test Generation. Master thesis, Software Engineering Research Group, Delft University of Technology, Delft, The Netherlands, 2010.
[11] CREST. http://code.google.com/p/crest.
[12] [Hamood, S. C Slicer. MSc Individual Project, Msc Advanced Software Engineering, Department of Computer Science, King's College London, University of London 2004 - 2005.
[13] Harman, M. ,and Danicic, S.. Using Program Slicing to Simplify Testing. Journal of Software Testing, Veri_cation and Reliability, 5(3), pages 143-162, 1995.
[14] Chen, Zhenqiang and Xu, Baowen and Yang, Hongji and Chen, Huowang. Test coverage analysis based on program slicing. IEEE

International Conference on Information Reuse and Integration, 2003. , pages 559-565, 2003.

[15] Z. Awedikian, K. Ayari, and G. Antoniol. MC/DC automatic test input data generation. In Proc. GECCO, pages 1657-1664, New York, USA, 2009.

[16] SangharatnaGodboley, G.S.Prashanth, Durga Prasad Mohapatra and BansidharMajhi."Increase in Modified Condition/Decision Coverage Using Program Code Transformer ", In proceedings of 2013 3rd IEEE International Advance Computing Conference (IACC), Gaziyabad(U.P), Pages: 1401-1408, Feb 2013.

[17] SangharatnaGodboley, SaiPrashanth, Durga Prasad Mohapatra and BansidharMajhi. "Enhanced Modified Condition/Decision Coverage Using Exclusive-NOR Code Transformer", In proceedings of 2013 IEEE International Multi Conference on Automation, Computing, Control, Communication and Compressed Sensing (IMAC4S), Kottayam, India, 22nd - 23rd March 2013.

The author name is **Prof. Sangharatna Godboley** and was born on 1st July 1990 at Nagpur Maharastra. The author completed his B.E degree from Government Engineering College Bilaspur, affilited to CSVTU Bhilai University. He did his M.Tech degree from National Institute of Technology Rourkela under the guidance of Prof. D. P Mohapatra and Prof. B. Majhi. He is a IEEE student member and IEEE Communication Society member since1st Jan 2013. He has published two IEEE international conferences and one Springer conference. He communicated with FIVE more research paper.

AVIJIT DAS was born in Patna, India on 1st January 1983. He received his B.Tech degree from NIT Silchar and his M.Tech degree in Computer Science and Engineering from IIT Kharagpur. Currently, he is holding the position of Scientist-D in Defence Research & Development Organisation, India. He is working in the area of independent verification and validation of avionics software. During his M.Tech course he worked under the guidance of Professor Rajib Mall for his M.Tech thesis and published a paper in automatic mc/dc test data generation in an international journal.

Kuleshwar Sahu born on 21st March 1989 at durg, India. He did his B.E from Government Engineering College Bilaspur, affilited to CSVTU Bhilai university. He did his M.Tech degree from National Institute of Technology Kurukshetra. He is Software Developer at TCS Ahmadabad India. He has published one Springer conference and one International conference paper. His areas of interests are Data Mining and Advanced DBMS.

Prof. Durga Prasad Mohapatra received his Ph. D. from Indian Institute of Technology Kharagpur and M. E. from Regional Engineering College (now NIT), Rourkela. He joined the faculty of the Department of Computer Science and Engineering at the National Institute of Technology, Rourkela in 1996, where he is now Associate Professor. His research interests include software engineering, real-time systems, discrete mathematics and distributed computing and published more than forty papers in these fields. He has received many awards including Young Scientist Award for the year 2006 by Orissa Bigyan Academy, Prof. K. Arumugam award for innovative research for the year 2009 and Maharasthra State National Award for outstanding research for the year 2010 by ISTE, New Delhi. He has also received three research projects from DST and UGC. Currently, he is a member of IEEE. Dr. Mohapatra has co-authored the book Elements of Discrete Mathematics: A computer Oriented Approach published by Tata Mc-GrawHill.Computer Science and Engineering Dept., National Institute of Technology, Rourkela, India.

Prof. Bansidhar Majhi received his Ph. D and M. E. from Regional Engineering College (now NIT), Rourkela. He joined the faculty of the Department of Computer Science and Engineering at the National Institute of Technology, Rourkela in 1991, where he is now Professor. His research interests include Soft Computing, Image processing, Biometrics, Security Protocols. He is a member of professional bodies like FIETE, LMCSI, and AMIE (INDIA). Computer Science and Engineering Dept., National Institute of Technology, Rourkela, India.