Automatic Test Case Generation of C Program Using CFG

Sangeeta Tanwer and Dr. Dharmender Kumar

**Computer Science & Engineering, Guru Jambheshwar University of Science& Technology
Hisar, Haryana, India**

Abstract

Software quality and assurance in a software company is the only way to gain the customer confidence by removing all possible errors. It can be done by automatic test case generation. Taking popularly C programs as tests object, this paper explores how to create CFG of a C program and generate automatic Test Cases. It explores the feasibility and non-feasibility of path basis upon no. of iteration. First C is code converted to instrumented code. Then test cases are generated by using Symbolic Testing and random Testing. System is developed by using C#.net in Visual Studio 2008. In addition some future research directions are also explored.

Keywords: *software testing; random testing; symbolic testing; test case generation; Path feasibility.*

1. Introduction

In the industry, test cases are generated manually. This is a very slow process and the human involvement in this process leads to the involvement of human biases. Ultimately, the result is the generation of ineffective and inadequate test cases. Finally, the quality of the software is affected [5].

Automatic Test Case Generation Tool is the answer to all the issues discussed earlier [9]. This test case generation tool develops test cases for a 'C' language program. It automates the unit testing of the software by generating test cases for a program / unit. This automatic test case generation tool use the structural testing [11, 12] and symbolic testing to generate the test cases. The random testing is also be used in this tool to randomly generate values that would be used for identifying the test cases.

2. System Process

This automatic test case generation is developed using C#.net framework on window platform.

Various steps for test case are given as: (1) Automatic test case generation tool would be an automated test case generation system that allows the tester/user to input a 'C' program. It will parse through the program. Based on the internal structure of the program, CFG (Control Flow Graph) for the input program is generated. This will provide a graphical view of the logic of the program. (2) The automatic test case generation tool will use this CFG generated to generate the CFG matrix. Using this CFG matrix, various possible paths through the program will be identified. (3) First, the automatic test case generation tool will be implemented using random testing based technique, then it will allow up to some extent to get the information about the feasibility or infeasibility of a path / difficulty in solving the path. Finally, this tool will generate the test cases for input program. (4) The automatic test case generation tool will develop test cases based on best available testing criteria that will provide the assurance of complete possible testing and also provide the information about the termination of the testing process. Unit testing is used for this research by analysis of all testing strategies [14].

3. Path Testing

Path Testing is a testing technique in which a set of paths are selected from the domain of all possible paths through the program [3].

Consider one example:

if(abc<0)

{

In this part, there is no definition of any variable.

}

And the else part does not exist.

It is analyzed to use path testing for the test case generation [8].

Next issue is about testing criteria.

Suitable test criteria can be the way to divide the program input domain into a path. The *path coverage* is the strongest criteria in the path testing family [1, 14].

4. Automatic Test Case Generation Tool

For automatic test case generation, Computer Based Testing technique - Symbolic Execution is used. Symbolic Execution is performed using Random Testing Based Automatic Test Data Generation technique, for this Automatic Test Case Generation Tool.

5. CFG

CFG stands for Control Flow Graph. It describes the logical structure of the program. It consists of nodes and edges. Actually it is a directed graph which shows the all possible ways [2,6] for the flow of control through the program beginning from the start node to the exit node of the program. These various possible flows of control through the program are the various possible paths through the program. A CFG function is created which is having a class field *collate* and *resources* field resource Culture and *resource Man*. It has a function *put Braces InIfElse* to put braces in if else structure in proper format for CFG. Other function *loop To IfFormat* is used to convert loop in if statement for creating graph. loopConversionResult to convert in return conversion for loop.

Consider one simple example to better understand the concept of CFG.

```
1-  #include<stdio.h>
2-  void main()
3- {
4- int a,b,c,flag;
5-printf("enter three sides of a
triangle");
6-scanf("%d%d%d",&a,&b,&c);
7- printf("Side A is : %d", a);
8- printf("Side B is : %d", b);
9- printf("Side C is : %d", c);
10-if((a<b+c)   &&   (b<a+c)   &&
(c<a+b))
11- flag=1;
12- else
13- flag=0;
14-if(flag)
15-
16-if((a==b) && (b==c))
17-printf("equilateral
triangle");
18-else
19-if  ((a!  =b)  &&  (a!=c)  &&
(b!=c))
20-printf("scalene");
21-else
22-printf("isosceles");
23-}
24-else
25-printf ("not triangle");
26-printf ("end of program");
```

Refer Fig. 1 for the CFG of program mentioned above.

In *Fig 1*, Node with label 5-9 represents the accumulation of statements no 5, 6, 7, 8, 9 at a single node as they all are sequential. The non-executable statements like variables and type declarations are not considered in CFG. All the decision making statements and looping statements are shown by separate nodes.

As shown in the figure, labels T & F at the decision making nodes outgoing edges represents the flow control from one node to the other node in the graph based on the fact whether the condition on decision making node is true or false.

5.1 Instrument

It is used to convert the c language code in the instrumented code. In this instrument Source Code function is used to avoid all comments [4] and avoiding initial statements & storing # define values in array List. Defined symbolic constant replaced with corresponding values.

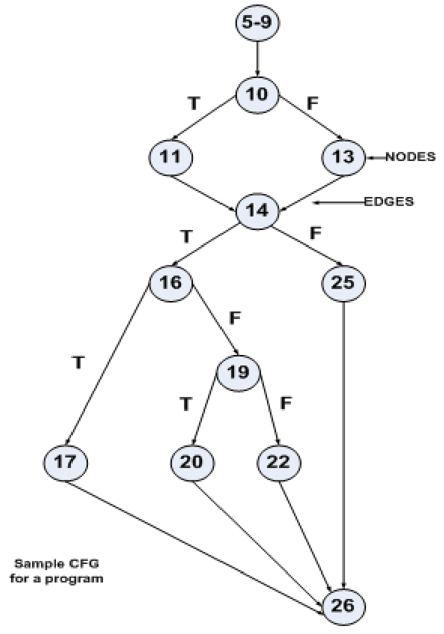

Figure 1

5.2 CFG Matrix

CFG Matrix is the *adjacency matrix* of the Control Flow Graph (directed graph).

What are feasible paths?

Feasible paths are the paths for which some input data is available to execute them and for infeasible paths vice-versa.

6. Symbolic Execution

This technique can be used to test a program by computing full symbolic output values or it can also be used to generate test cases for a program.

During the symbolic execution of a program, the actual data values are replaced by symbolic values. In symbolic execution, the input variables of a program are assigned symbolic values. These symbolic values are fixed and unknown. Symbolic execution basically distinguishes between two types of statements, one is the assignment statement and the other is decision making / branching statement.

Refer Fig. 2 to have a clear understanding. At node 1, the condition for decision making/ branching is (A>0 & B>0 & C>0). If (A>0 & B>0 & C>0) yields true then, the control flow of the program will move from node 1 to node 3 and if (A>0 & B>0 & C>0) yields false then, the control flow of the program will move from node 1 to node 2.

So, during symbolic execution, the predicate expression for case (when condition (A>0 & B>0 & C>0) is true) is expressed as (A>0 & B>0 & C>0) and in other case (when condition (A>0 & B>0 & C>0) is false), the predicate expression will be !(A>0 & B>0 & C>0).

Similarly, it will be done for each decision making / branching statement.

Finally, for a particular path, all the predicates expressions are conjunctively joined to form path predicate expression. This path predicate expression will contain the input and internal variables in terms of symbolic values.

Suppose if path: Node1-> Node3-> Node4 ->Node14 is considered. Then, the symbolic evaluation of this path yields the following *path predicate expression:*

((A>0 && B>0 && C>0) AND !(A<B+C && B<A+C && C<A+B)).

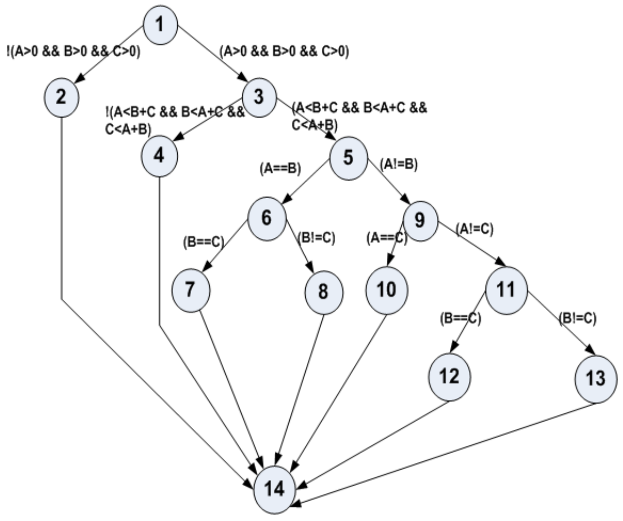

Figure 2

Now, using random testing the symbolic values are placed by randomly generated values and the values which will satisfy this path predicate expression, means those set of values can execute the path and the set of values is one test data for the program.

6.1 Random Testing Technique

Random Testing Technique is used in this automatic test case generation tool to select values randomly from the input domain of the program input variables. These randomly generated values are assigned to the symbolic values and then the predicate expressions for all decision making nodes in a path are joined using logical AND to form a conjunctive clause for that particular path. Then, randomly generated values are used to satisfy the conjunctive clause. The values which will be able to satisfy the conjunctive clause for a path become the test data for that path to execute.

7. Conclusion and future work

During the Whole life cycle of software testing plays an important role. There are various issues related to the instrumentation of input program and symbolic testing of the program in the testing framework. Instrumentation of input program and symbolic execution of the program are necessary steps to generate test cases. In the testing framework, this research identifies various problems and issues and provides solutions to overcome these problems and to handle the related issues. In the future it is possible to remove all errors and improve more the quality of software by using Genetic algorithm technique to use static technique for test case generation.

Acknowledgement

This work was supported by Dr. Dharmender Kumar. I would like to thanks to him for the continual encouragement, help and guidance throughout the work.

References

[1]. Andreas S. Andreou "An automatic test data generation scheme based on data flow criteria and genetic algorithms "*Third International Conference on natural Computation(ICNC)* Vol 1, pp 2, 2007.

[2]. Bruce A. Cota "Control flow graph as representation language" *Winter Simulation Conference,* pp 556-559,1994.

[3]. Chao-Jung Hsu "Integrating path testing with software reliability estimation using control flow graph" Management of Innovation and Technology Forth IEEE International Conference CMIT pp. 1234-1239,2008.

[4]. Chengying Mao,Yansheng Lu "Cpp Test: A Prototype Tool For Testing C++" *Second International Conference on Availability and Security(ARES'07,2007*.

[5]. Carlos Urias Munoz "An approach to software Product testing" IEEE *Transaction on Software Engineering, Vol 14, NO I I , November 1988* .

[6]. Douglas G. Fritz "An overview of hierarchical control flow graph models" *Proceedings of the 1995 Winter Simulation Conference* pp 1347-1355, 1995.

[7]. DeMillo, R.A.anmd Offutt, A.J "Constraint-Based Automatic Test Data Generation", *IEEE Transaction on Software Engineering* Vol17,No 9,September pp 900-910, 1991.

[8]. Howden,W.E. "Reliability of the Path Analysis Testing Strategy", *IEEE*

[9]. Jon Edvardsson "A Survey on automatic test data generation", *Second Conference on Computer science and Engineering in Linkoping,* Vol 23 ,pp 21-281, 1999.

[10]. Miller, W. and Spooner, D.L. "Automatic Generation of Floating-point Test data", *IEEE Transactions on Software Engineering* Vol. SE-2,No.3 ,pp 223-226,1976 .

[11]. Nigel Tracey John Clark Keith Mander John McDermid **"**An automatic framework for structural test data generation" *IEEE* 2007.

[12]. Ntafos, S.C. "A Comparison of Some structural Testing Strategies", *IEEE Transactions on Software Engineering* Vol 14 No. 6, pp 868-873, 1988.

[13]. Patricia Mouy "Generation Of All Path unit test with functional calls " *International Conference on Software Testing, Verification, and Validation* pp 32-41, 2008

[14]. Weyuker,E"Axoimatizing software test data adequacy", *IEEE Transactions on Software* Vol. 15 N0. 4 ,1989.