Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 62 (2015) 266 – 273

Procedia Computer Science

The 2015 International Conference on Soft Computing and Software Engineering (SCSE 2015)

ABCE: A novel framework for Improved Branch Coverage Analysis

Sangharatna Godboley[a], Arun Sahani[b], Durga Prasad Mohapatra[c]

DOS Lab NIT Rourkela, Rourkela, Odisha, India [a,b,c]

Abstract

The software testing practices generate test cases manually, that affects both the effectiveness and efficiency. In aerospace and safety critical domains, software quality assurance is strict to rules and regulations such as DO-178B standard. To resolve these issues, concolic testing generates test suite that can attain high coverage using an automated technique based on Branch Coverage. In this paper, we propose a framework to compute branch coverage percentage for test case generation. To achieve an increase in branch coverage, we transform the input Java program, P, into its transformed version, P', using JPCT. Then we use JCUTE to generate test cases and find branch coverage percentage. Our experimental studies on branch coverage percentage consists of two steps: the first observation is made without using Java program code transformer, the second observation is made by using Java program code transformer. The Java program code transformer adds additional conditional statements for each decision that causes variation in branch coverage. We quantify this variation in the form of branch coverage percentage. This approach resolves some of the bottleneck issues associated with traditional concolic testers. The experimental studies show that our approach achieves 20.146 % of average increase in branch coverage. This increase in branch coverage percentage is achieved in an average computation time of 33837.6 milli seconds.

Keywords: Software Testing; branch coverage; JPCT; JCUTE;

1. Introduction

To improve the software quality, software testing targets to detect and fix all the critical bugs in a software. Automated software testing saves about 40% to 50% of the software development efforts. Earlier days testing methods however, often fail to detect bugs in the softwares. One of the reasons being that a program may have enormous different execution paths because of the presense of loops and conditional statements. Therefore, it is unlikely for a test engineer to manually create enough test cases to detect subtle bugs in all the possible execution paths. It is technically a big challenge to generate a test suite that covers all different paths in an automated manner.

* Sangharatna Godboley. Mobile Number.: +91-9040504120 ;
E-mail address: sanghu1790@gmail.com

To address these issues, CONCOLIC (CONCrete + symbOLIC)[2] testing combines concrete dynamic and static symbolic analysis to automatically generate test cases for exploring execution paths of a code and achieve high coverage[9 10]. Where as, concolic testing may take a significant amount of time to explore all the execution paths, and this is an issue towards its practical application[3].

There exists various types of structural coverage criteria such as statement coverage, condition coverage, branch coverage, modified condition/decision coverage (MC/DC), and multiple condition coverage (MCC). MC/DC subsumes all other previously mentioned structural coverage criteria but MCC. MC/DC is a very strong coverage criterion in detection of common bugs. The process of manually achieving coverage is both very difficult and effort intensive. In accordance to DO-178B standard, coverage based testing is a measure of acceptability of the requirement-based testing in the context of exercising logical expressions.

In this paper, we propose a framework to calculate the branch coverage percentage of Java programs. We have named our framework Architectural model for branch coverage Enhancement (ABCE). ABCE consists of mainly two modules: i) *Java Program Code Transformer (JPCT)*[7] , ii) *Java Concolic Tester (JCUTE)*. In this approach, we designed JPCT in ABCE to improve the effectiveness and efficiency of the proposed framework. Our framework ABCE overcomes the problems of traditional concolic testing. JCUTE takes the non-transformed program and the transformed program as input and generates the test cases and calculates the branch coverage percentage.

The rest of the paper is organized as follows: Section 2 discusses some of the fundamental ideas required to understand the proposed work. In Section 3, we discuss some of the existing related works for our proposed approach. Section 4 introduces the proposed ABCE framework and discusses the required steps for implementation. We give the analysis of our experimental results in Section 5 and report some of the existing threats to the validity of our approach in Section 6. Section 7 concludes with some insights to our future work.

2. Fundamental Ideas

Basically our proposed approach is based on Branch Coverage and CONCOLIC testing. Our input programs must have predicates to process. To have more information regarding this process, before presenting our approach we discuss some basic but important definitions such as *Condition*, *Decision*, *Group of Condition*, *Branch Coverage*, and *CONCOLIC testing*. Below we discuss some fundamental concepts required to understand our approach.

Condition:A condition or a clause is a Boolean Expression without any Boolean Operator.
Decision:A decision is a Boolean Expression composed of one or more conditions within a decision.
Group of Conditions:A decision or predicate is composed of two or more conditions with many Boolean Operators.
Branch coverage: Each decision should take all possible outcomes at least once either true or false[1]. In this coverage, the test cases are (1) m=n, (2) m! =n, (3) m>n, (4) m<n.
Concolic Testing: The concept of CONCOLIC testing combines a the CONCrete constraints execution and symBOLIC constraints execution to automatically generate test cases for full path coverage this testing genertae test suites by executing the program with random values. At execution time both concrete and symbolic values are saved for execution path. The next iteration of the process forces the selection of different paths. The tester selects a value from the path constraints and negates the values to create a new path value. Then the tester finds concrete constraints to satisfy the new path values. These constraints are inputs for all next executions. This process is performed iteratively until the covered branches exceed the threshold value or sufficient code coverage is obtained[9 11].

3. Related Works

In this section we discuss our litreture survey in detail. We compare the different work on concolic and coverage based testing Table 1. Table 1 compares the different aspects (testing type, framework type, input type and output type)of some available work[3 9 4 8]. We did our survey with other twelve related works. Table 2 represents characteristics of the different approaches such as test cases, coverage type, and time constraints.

Godboley et al.[4] proposed the approach of program code transformation technique for C programs to measure MC/DC %. They used concolic tester CREST to generate test cases and designed coverage analyser to measure percentage. In our approach we are measuring branch coverage percentage for Java program using JCUTE. We