Summary
Detection of Infeasible Paths through Recognizing their Patterns

Hee Beng Kuan Tan, and Minh Ngoc Ngo
School of Electrical and Electronic Engineering
Block S2
Nanyang Technological University
Nanyang Avenue
Singapore 639798

Hongyu Zhang
School of Software, Tsinghua University
Beijing 100084, China
hongyu@tsinghua.edu.cn

1. Background

In testing a software system, code logic path coverage -- the percentage of code logic paths exercised by a test suite -- is a very important measure for the adequacy of the test suite. Unfortunately, all existing software testing tools are unable to compute such measurement to a useful level of accuracy due to the following reason:

> Many logic paths in code are infeasible – there is no input for which these paths will be executed. Theoretically, the detection of these infeasible logic paths is an unsolvable problem. Existing methods for detecting infeasible paths apply symbolic evaluation directly. This results to extensive symbolic evaluation that incurs heavy computation. Moreover, none of the techniques can detect more than 50% of the infeasible paths.

2. Novelty of the Invention

Though theoretically, there is an infinitely number of forms that infeasible paths may take, empirically, we have found that a lot of theoretically possible forms rarely occur due to the fact that no programmers write codes in those manners. The consideration of such human factor, however, has not yet been explored in the existing methods for detecting infeasible paths.

With the consideration of the above-mentioned human factor, our empirically investigation have discovered that almost all infeasible paths follow a few patterns. Based on this, we establish a novel method for detecting infeasible paths. The unique novel feature of our method is:

To determine whether a path is infeasible, we first check whether a path may follow any of the above-mentioned patterns using solely static program analysis. If it is not possible for a path to follow any of these patterns, we conclude that the path is not infeasible without any symbolic evaluation. Only if a path may follow one of the patterns, simple symbolic evaluation is then applied with the guide of the pattern.

In our method, experimental results shows that the infeasibility of a larger proportion of paths can be determine solely using static program analysis to check against the above-mentioned few patterns. Symbolic evaluation is only required for a smaller proportion of the paths. And, the symbolic evaluation involves only a few nodes and therefore much simpler than general symbolic evaluation on predicate nodes. This shows that our method avoids heavy computation. More importantly, the experimental result also shows that our method can detect all infeasible paths.

3. Advantages and Improvement over Existing Methods

The advantages and improvement of our method over exiting methods are as follows:

1) Our method can detect almost all infeasible paths
2) Our method avoids heavy computation

Our extensive experiments that include open-source and industry codes have provided conclusive evidence on these advantages. Hence, our method enables accurate detection of infeasible paths to be implemented in software testing tools for path-based coverage analysis.

4. Technical Description of the Invention

In our method, the process of detecting infeasible paths can be divided into two stages:

1) Target paths are checked against the above-mentioned few patterns that we have discovered. Paths that will not fall under any of these patterns can be concluded as not infeasible right away.
2) For those paths that may fall under one of the above-mentioned few patterns, nodes in a path are extracted according to the pattern and simple symbolic evaluation is carried out to evaluate its infeasibility.

5. Commercial Application of the Invention

Software Testing takes up more than 40% of the total effort and cost in Software Development. Software Testing tools have a large and rapidly growing market. Direct important commercial applications of the invention are:

1) To implement the proposed method in Software Testing tools to enable them to compute the coverage of logic path in code accurately and therefore provide a meaningful and useful measure to show the adequacy of a test suite. This will lead to a much accurate way to certify the completion of testing.
2) To implement it in test case design and generation technique to reduce those input conditions that are infeasible and therefore reduce the test requirements.

The method is also useful in code optimization, program slicing, and other program analysis to improve their efficiency and precision.