Automated Test Data Generation and Reliability Assessment for Software in High Assurance Systems

Branson W. Murrill

Department of Computer Science
Virginia Commonwealth University
bmurrill@vcu.edu

Abstract

Validation of software is often a weak link in delivering high assurance systems. Traditional software testing is problematic in both the selection of good test data and the assessment of reliability after testing.

This paper describes ongoing research in perturbation analysis, which is a dynamic software analysis technique based on perturbing the data state at various locations in an executing program. The results of perturbation analysis have many uses, including the automatic generation of robust test data and reliability estimates for code locations and execution paths. An experiment is described that applies perturbation analysis to a software component of the Tactical Tomahawk Weapons Control System. The application of perturbation analysis to produce reliability estimates for software components is also discussed.

1. Introduction

Verifying that software components perform their intended function and quantifying the results of such verification is critical to the delivery of high assurance systems in life-critical and business-critical applications. Traditional software testing techniques are often not sufficient for this purpose. Functional tests are usually selected manually by the tester to demonstrate correct operation of a specified function; such tests are time-consuming to construct and prone to human error. Coverage based testing, like control flow and data flow strategies, can be automated with the help of tools but do not enable reliability estimates after the successful conclusion of testing. None of these testing techniques fully exercise the semantics of the code.

Perturbation analysis is a dynamic technique used to analyze the semantic behavior of software when data state changes (perturbations) are made in an executing program. The results of perturbation analysis can be used generate robust test data and estimate the reliability of code after testing. [1,2]

Section 2 gives a brief background on perturbation analysis. Section 3 describes an experiment that applies perturbation analysis to a component in a real-time control system. Section 4 discusses how the results of perturbation analysis can be used to produce reliability estimates for software components.

2. Perturbation analysis

A program can be viewed as a function that maps an input domain into an output domain. A data state, consisting of all memory locations that can be modified by the program’s execution, exists before and after each program instruction. The instruction’s function upon execution maps the previous data state into a subsequent data state. The execution of a path in the program is a composition of the instruction functions along that path.

The manifestation of a code fault as a failure requires 1) that the fault be executed, 2) that the execution introduce an error (infection) into the data state, and 3) that the infection propagate to output. Perturbation analysis is similar to mutation analysis in that it injects a change into a program at a given location and determines if that change affects the output. Mutation analysis uses the syntactic modification of a statement to produce the change while perturbation analysis directly changes the value in a location’s subsequent data state. A test input that yields a different output from that of the original program is sufficient to detect the mutation or perturbation. A program can be instrumented to produce perturbations at the primitive operation level (add, move, etc.) making execution and propagation the only issues in determining the effect of the perturbation on the code’s behavior. Performing the
analysis for many perturbations at a location enables estimation of the propensity of the remaining code on that path to reveal a fault. It also allows the capture of inputs that will reveal those perturbations. This analysis is extended to all locations in the code, and the results can be used to produce a robust (perturbation-adequate) test set and estimate execution and propagation rates for locations and paths.

3. A perturbation experiment

An experiment is reported in [3] that applied perturbation analysis to a software component of the Tactical Tomahawk Weapons Control System (TTWCS), developed at Dahlgren Naval Surface Warfare Center. The component of TTWCS investigated is the Cell Pre-selection component, which automates the process of allocating Tomahawk missiles carried aboard a submarine to a strike plan. The algorithm selects onboard missiles based on their capabilities by optimizing a set of weighted criteria relative to a set of mandatory constraints. This function was first implemented as a prototype written in Java, consisting of about 6500 source lines in more than 50 class modules, and tested with unclassified data. Based on the experience gained from the prototype, the deployed component was coded in C++ to run on dedicated shipboard computers. Although not a direct porting of code, the functionality between the prototype and deployed code is close enough to be able to use test data generated for the prototype. Two hand-generated (at significant time and expense) test sets were developed to validate the correct functionality of the allocation algorithm in the deployed code.

The prototype code was hand-instrumented for perturbation analysis and run on a Windows-based laptop. The perturbation analysis done was only partial, focusing on integer and boolean variables in those classes that directly managed the missile allocation algorithm. The goal was to develop test sets that would detect 99% of the perturbations generated.

Early experimentation revealed that a number of locations never had detectable perturbations. Upon review with the prototype’s developer, it was determined that much of this was due to redundant or obsolete code, making this a useful tool for “code cleanup”. After removing those variables from the perturbed set, the process of generating 99%-adequate test sets continued. This building process was fully automated but computationally intensive, with the time to develop each set ranging from 30 to 70 hours of computer time. It should be noted that there is much room for performance improvement in developing future perturbation analysis systems. The hand-generated test sets for the deployed code were analyzed for perturbation adequacy and achieved 72% and 73% detection. However, these test sets were only designed to verify the algorithm, not cover all of the code.

The test data derived from perturbation analysis of the Java prototype code was used to test the actual C++ deployed code. Taking into account the functional modifications designed into the deployed code, the derived tests performed similarly to the hand-generated tests in exercising the missile allocation function and producing valid allocations. Interestingly, path coverage analysis of the deployed code showed that the derived test set (30 test inputs) covered 341 of 484 possible paths (70.5%) while the smaller of the hand-generated test sets (38 test inputs) covered 286 paths (59.1%). Thus, the automatically generated perturbation-adequate test set provided greater code coverage with fewer test inputs than the manually generated test set.

4. Using perturbation analysis to estimate reliability

Perturbation analysis provides path propagation rate estimates over all locations covered by each path in the program. These estimates can be used with the distribution of test inputs among paths to find the probability that a program with a fault at a given location would fail under testing. [1] Alternatively, test inputs can be added to drive paths that increase this probability for every location up to some common value, like .99999.

5. References

