Random Generation of Test Inputs for Model Based Unit Testing of Object-Oriented Programs

Devadatta Sadhu, Aritra Bandopadhyay, Sudipto Ghosh, Robert France
Department of Computer Science
Colorado State University
Fort Collins, Colorado 80523
{sadhu, baritra, ghosh, france}@cs.colostate.edu

ABSTRACT:
Randomized and automated test input generation for building unit tests for an object-oriented system involves the creation of random test objects that are compliant with the system's class model. Existing approaches to random test generation, such as RANDOOP, maintain a pool of previously generated valid method call sequences and use them to produce new valid test inputs, whereas invalid sequences are discarded. However, this approach does not account for higher coverage of instances of the classes in the class model of the system under test. In our approach, we generate test inputs which are object configurations of the given class diagram. We intend to generate valid test inputs with adequate coverage of the domain of these object configurations. Each time we generate new object instances of these classes with random configurations, i.e., we do not extend previously generated valid configurations. We check the configuration for conformance. By conformance, we mean the object configuration of a valid test input must satisfy the constraints specified in the program's class model. If the generated configuration does not satisfy the class diagram constraints, we modify the inconsistencies to make it consistent. Instead of discarding this invalid configuration, we fix the non-conforming issues and generate a valid test input. An invalid configuration can be fixed by implementing one or more of the following techniques:

a) Add links, if observed number of links between two object instances is less than the specified number
b) Add object instance, if an object instance of a class is not created
c) Remove links, if observed number of links between two object instances is greater than the specified number
d) Remove object instance, if an object instance is not referenced by any other object

In our preliminary evaluation, we observed our test cases detected more faults compared to the RANDOOP test cases. Our next step is to generate more efficient and effective test cases in terms of domain coverage of the object instances. An invalid configuration can be modified by implementing either or more of the above mentioned techniques. We want to find out the path for which minimum number of fixing needs to be done. At the same time, we want to generate more instances of the object configurations. Our goal is to generate more instances of object configurations with reduced number of fixing to make those configurations valid. These two aspects of our goal apparently seem to be conflicting. To address these two aspects, we will develop a search-based approach that will provide an optimal solution to our problem.

***Keywords*:** UML class diagram, random testing, object configurations, constraints, violating instances