Relational Symbolic Execution of SQL code for Unit Testing of Database Programs

Symbolic Execution executes the unit test over symbolic input values instead of concrete ones. It is utilized as a white box testing method, to ensure a large portion of the program is executed. In simple terms, Symbolic execution works to make test cases that can encompass large amounts of the programs, specifically if there are logical branches that may not be utilized in a single run. This utilized under the weight of unit testing, means to apply runs on different cases on specific modules to span a large amount of that module. As well, classical code mixed with SQL, like PHP, has always found a need for testing to prevent data corruption. Thus, a marriage between Symbolic Execution of unit testing in the presence of SQL statement seems like a natural mix. Unfortunately, the reality of coding constraints makes this task very complex. Deciding the satisfiability of an SQL query is not computationally possible, why use this specific query instead of another if they both output the same result. Transferring these test cases into the SQL becomes a difficult endeavour. “SQL is a declarative language…They do not make explicit the sequence of operations necessary to compute this action.” To overcome this obstacle many researchers have introduces native variables in the classical code to represent the database content and replace the SQL queries and modifications by native code acting alongside the new variables. Let’s say we have java for example, instead of including the embedded SQL commands, they will be Java. This way, the auto generation of test scripts alongside SQL can be of much ease while still ensuring the test walks through all key paths and reduces the chance of bugs significantly in the module.

This task is a very interesting problem. On one hand, you have a very formidable software in the symbolic execution. This meeting the SQL head on, a language that doesn’t have a proper way to follow its structure and spit out test cases. The work around, is to create new variable within the original code that work as directives, building and querying a database without leaving the comfortable confinements of the native language. This way the symbolic execution can continue to create test cases, by looking at the different paths inside of the original source code. It can continue build solid test scripts to ensure these modules are tested properly and ready for integrating with the rest of the code. This form of unit testing is implemented through Whitebox methodologies and can work well with backend frameworks in web and app development, that utilize databases to a heavy extent.

Pythia reloaded: an intelligent unit testing-based code grader for education

The paper proposes a unique coupling between unit testing frameworks and automated competition software graders for automatic assessment of educational code. Seeing as assessment of code in educational programs, normally is provided in small predefined modules with set parameter. It creates an environment where unit testing can succeed. Currently, two major kinds of code assessment tools exist: unit testing frameworks and competition graders. Though each have their own issues working alone with automated assessment tools, Unit testing just outputs the testcases that failed and the ones that passed. This does not work well with learners, who need a more in-depth explanation to their results. The issue with competition graders is “that they are very specific to satisfy the constraints they have to meet. Therefore, they often only support a single programming language.” Although both these tools have their perks, they both carry key issues as standalone automatic assessments. Therefore, Pythia was designed. “The platform combines a unit testing framework with a competition grader.” This way the competition graders can be utilized for the isolated sandbox for the safe execution of code. All the while, the unit testing framework brings a structured way to test the learning code. To add, the platform adds a feedback system that can give accurate responses based on why the test case failed. A key fault with unit testing in the learning environment is resolved here.

This opens a new way of looking at testing code. As developers, we work with many unit testing frameworks. Now this testing framework needs to work at a higher level. Directly in the hands of the clients, this testing tool needs to work not too help make a working software, but as the working software the consumer utilizes. The way Pythia helps to learning environments, by creating an intelligent feedback system coupled with the two frameworks shows the work that goes into creating a strong learning environment.

# Unit testing performance with Stochastic Performance Logic

# Unit testing works as a quality management tool in software development. Unfortunately, practical constraints make unit testing difficult to be utilized for performance testing. Therefore, Stochastic Performance logic was developed, “a formalism for expressing performance requirements, together with interpretations that facilitate performance evaluation in the unit test context.” The small scope of unit testing makes it flexible, where little installation is necessary. This makes unit testing an attractive take for performance testing. There are various hurdles that need to be jumped through, including test implementation, and it design which can lead to many misleading results. A framework is designed to overcome these hurdles as well as succeed as a performance unit test framework. Central to the framework is Stochastic Performance Logic. This mathematical language is used to express requirements to better work alongside the unit test parameters.

# This SPL based performance testing has been integrated into an automated testing framework for Java, akin to JUnit. An example of the successful result can be seen by the 58 tests in where 6 cases were utilized to catch many bugs in the performance, bugs not caught through the process testing. This new testing software has seen success and revolutionary work though its research to design a formalized math language to help create the process unit test frameworks. I believe this, of the test cases ive worked on, has the potential to make for many greater findings. The utilization of math and science to build this framework not only helps people in the industry, but I believe people in academia can look at this tool for further understanding and greater knowledge.

# <https://link.springer.com/article/10.1007/s10515-015-0188-0>