Philipps-Universität Marburg
Fachbereich Mathematik und Informatik
AG Programmiersprachen und -werkzeuge

Classification and Recognition of Mutations in Code

Motivation

Mutation-Testing (MT) is a technique to assess the strength of a test suite for application code in a software project. Assuming that the test suite is satisfied in the beginning, the MT tool *mutates* the application code, i.e, makes a small change like replacing a plus with a minus (which is called a *mutation operator*), and runs the test suite again; this is repeated for a large number of mutations. Since the mutation has changed the application's behavior, each change should be detected in terms of at least one failing test. If this is not the case, this is a sign that the test suite is not strong enough and therefore would also miss actual mistakes in the program. Several MT tools exist, primarily in research, which follow different approaches for implementing mutation operators, making it difficult to compare these tools. In most cases, the tools themselves report which mutation operators have been applied in a specific run, but due to different design philosophies in these tools, the reports are not directly comparable. For example, one tool may report that is has replaced one arithmetic operator with another, while a different tool may report to have replaced `+` with `-`, `+` with `*`, `+` with `/`, and so on. Also the tools have different ways of reporting the location at which the mutation operator has been applied. Lastly, some of the tools mutate (Java) source code while others mutate at the (Java) bytecode level, which also makes a comparison difficult.

Assignment

In this assignment, actual mutants generated by MT tools should be inspected and the changes identified. For this purpose, a large inventory of mutants can be provided that have been generated by multiple mutation-testing tools. This includes the original code of several software projects and the code of the generated mutants. In all cases, the code is provided as Java bytecode, which is more general since it can easily be derived from mutated source code. To compare the original code and the mutant, our Java-bytecode metamodel ModBEAM should be used. This provides a fully automated way of creating an EMF-based model from Java bytecode. When the models of both code versions are available, they can be compared with the standard tool *EMF Compare* to identify the changes between both versions. This comparison should be performed on a large number of the mutants in the inventory and a classification of the identified changes should be derived. The goal is to provide an MT-tool-independent way of labeling the applied mutation operators and the location to which mutation operators are applied.

Further Reading

- Bugra M. Yildiz, Christoph Bockisch, Arend Rensink, and Mehmet Aksit. An mde approach for modular program analyses. In *Companion Proceedings of Programming' 17*. ACM, 2017. `doi:10.1145/3079368.3079392`
- Christoph Bockisch, Daniel Neufeld, and Gabriele Taentzer. MMT: Mutation testing of java bytecode with model transformation. In *Proceedings MODELS '23: Companion Proceedings*. ACM. `https://uni-marburg.de/6zkY40`
- Homepage of the EMF Compare project. `https://projects.eclipse.org/projects/modeling.emfcompare`. Accessed: 2023-11-16

Info

Bytecode, Mutation-Testing

Java, EMF

Bachelor or Master Thesis

Theory ▰▱▱▱
Practice ▰▰▱▱

Contact

Prof. Dr. Christoph Bockisch

`bockisch@mathematik.uni-marburg.de`