μBert: Mutation Testing using Pre-Trained Language Models

Renzo Degiovanni and Mike Papadakis

SnT, University of Luxembourg, Luxembourg

Conference: Mutation Workshop at the International Conference on Software Testing, Verification and Validation ICST 2022, Valencia, Spain, April 4-13, 2022.
`https://orbilu.uni.lu/retrieve/91556/98008/codebert_mutation-5.pdf`

Mutation testing seeds faults using a predefined set of simple syntactic transformations, aka mutation operators, that are (typically) defined based on the grammar of the targeted programming language. As a result, mutation operators often alter the program semantics in ways that often lead to unnatural code (unnatural in the sense that the mutated code is unlikely to be produced by a competent programmer).

Such unnatural faults may not be convincing for developers as they might perceive them as unrealistic/uninteresting, thereby hindering the usability of the method. Additionally, the use of unnatural mutants may have actual impact on the guidance and assessment capabilities of mutation testing. This is because unnatural mutants often lead to exceptions, or segmentation faults, infinite loops and other trivial cases.

To deal with this issue, we propose forming mutants that are in some sense natural; meaning that the mutated code/statement follows the implicit rules, coding conventions and generally representativeness of the code produced by competent programmers. We define/capture this naturalness of mutants using language models trained on big code that learn (quantify) the occurrence of code tokens given their surrounding code.

We introduce μBert, a mutation testing tool that uses a pre-trained language model (CodeBERT) to generate mutants. This is done by masking a token from the expression given as input and using CodeBERT to predict it. For example, given the masked sequence `int a = <mask>;`, CodeBERT predicts that `0`, `1`, `b`, `2`, and `10` are the (five) most likely tokens/mutants to replace the masked one (ordered in descending order according to their score – likelihood). Thus, the mutants are generated by replacing the masked tokens with the predicted ones.

We evaluate μBert on 40 real faults from Defects4J and show that it can detect 27 out of the 40 faults, while the baseline (PiTest) detects 26 of them. We also show that μBert can be 2 times more cost-effective than PiTest, when the same number of mutants are analysed. Additionally, we evaluate the impact of μBert's mutants when used by program assertion inference techniques, and show that they can help in producing better specifications. Finally, we discuss about the quality and naturalness of some interesting mutants produced by μBert during our experimental evaluation.