Clustering of Functionally Equivalent problem variations for representing the similar semantic patterns

Kamaleshwar Panapakam
Advisor: Prof. Carlos R. Rivero

ABSTRACT

- The Number of students registering in the Massive Open Online Courses (MOOCs) have increased significantly over the past few years.
- Providing a personalized feedback is becoming more and more complicated due to a multitude of submissions and the limited number of available tutors.
- Traditional approaches like Unit Testing, Peer Evaluation and Software Verification tools although solve this problem to some extent does not help the novice users.
- This led to the development of Automated Feedback Systems where the tutor can provide a custom feedback to a code fragment.
- These Automated Feedback Systems mostly rely on syntactic similarities and it is not feasible to store each possible solution that might match with the new one.

OBJECTIVE

- This research address the problem of program variability and proposes a data driven solution for identifying the programs that are semantically equivalent to the newly submitted program.

PROPOSED APPROACH

- Dataset consists of ASTs information for the below problems that are extracted from Leetcode
  - Finding the Greatest Common Divisor of two numbers
  - Finding the complement of a given number N
  - Repeatedly adding the digits of a number N
  - Sum of prime numbers up to a given N value
  - Sum of odd numbers up to a given value N
- The dataset is incremental and the new submissions are added to the database if it does not exist already in it.

DATASET

- Parse all the Java programs into Abstract Syntax Trees.
- Flatten the ASTs for each program into a list and serialize it.
- Store the serialized ASTs as a hash key and the serialized program code as a value in a Redis hash data structure.
- Prepare a set of test cases and store them in a separate hash data structure in Redis.

MATCHING ALGORITHM

1. procedure FindEquivalence(C, T, E)
2. ContextSet ← Sentence(C)
3. for each Context c, do ∈ ContextSet
4. A ← ℐ(c, a)
5. if A = E then ADD(S, c)
6. end if
7. end for
8. if size of C == Prev then return
9. end if
10. for each Candidate β ∈ S do
11. SubTreeSet ← Sentence(β)
12. for each SubTree α ∈ SubTreeSet do
13. C ← GetContext(α)
14. FindEquivalence(C, T, E)

RESULTS

- Input:
- Pattern for the Selected Code Fragment
  "12, 55, 83, 42, 31, 83, 83, 39, 42, 44, 39, 42, 8, 60, 39, 59, 42, 34, *, 41, 42"

- Output:

CONCLUSIONS

- The proposed approach successfully extracted the programs that are functionally equivalent from a body of other programs.
- The limitation of this approach is that when match is performed for a program that only has a single line of code, the search returns all the implementations of the submitted program.
- The matching algorithm can be even improved by performing semantic preserving transformation on the programs in the knowledgebase.

References:

3. Code fragments are extracted from LeetCode.

Contact
Kamaleshwar Panapakam - kp2601@rit.edu