Automatic Grading of Programming Assignments: An Approach Based on Formal Semantics

ABSTRACT:

Programming assignment grading can be time consuming and error-prone if done manually. Existing tools generate feedback with failing test cases. However, this method is inefficient and the results are incomplete. In this paper, we present AUTOGRADER, a tool that automatically determines the correctness of programming assignments and provides counterexamples given a single reference implementation of the problem. Instead of counting the passed tests, our tool searches for semantically different execution paths between a student's submission and the reference implementation. If such a difference is found, the submission is deemed incorrect; otherwise, it is judged to be a correct solution. We use weakest preconditions and symbolic execution to capture the semantics of execution paths and detect potential path differences. AUTOGRADER is the first automated grading tool that relies on program semantics and generates feedback with counterexamples based on path deviations. It also reduces human efforts in writing test cases and makes the grading more complete. We implement AUTOGRADER and test its effectiveness and performance with real-world programming problems and student submissions collected from an online programming site. Our experiment reveals that there are no false negatives using our proposed method and we detected 11 errors of online platform judges.

EXISTING SYSTEM:

In existing system, Our research seeks to identify semantic-level differences in input programs that can lead to potential path deviations. In this section, we review related work on auto grading and discuss existing methods for detecting semantic differences in programs. There are two critical issues affecting test-based grading. First, it can be costly to construct high-quality test cases. In fact, providing a complete test suite for a programming problem may not be practically feasible because of limited resources. Another problem with test-driven grading is that some tools generate feedback using formal methods if instructors have reference solutions to a programming problem. An error model is utilized to provide more in-depth and comprehensive feedback. However, it is arguable that writing a complete set of reference specifications is less expensive than constructing test cases, especially when the problem can be solved in numerous ways. Error models also require additional effort to devise.

DISADVANTAGES:

- Can be time-consuming
- Error-prone if done manually.

PROPOSED SYSTEM :

We have proposed a formal-semantics-based approach for automatically grading programming assignments with a single correct reference implementation provided by the instructor. By searching for inputs that can lead to path deviations between a student's submission and the reference implementation, we can determine the correctness of the submission and provide counterexamples for an incorrect submission. We have implemented this novel approach using a tool called AUTOGRADER and evaluated our technique over a dataset containing 10,270 submissions collected from an online programming site. The experiment revealed that our proposed method yielded no false negatives, while the online programming site yielded 11 false negatives due to incomplete test suites Our method relies on detecting semantic differences between the reference implementation and student submissions. While program equivalence is generally undecidable, we propose AUTOGRADER, a tool that automatically provides real-time judgments with counterexamples for programming exercises in introductory programming courses. The solution is based on program analysis, and more specifically, the technique of differential semantic analysis With a single correct implementation as the reference solution, AUTOGRADER can "search" for the differences between the execution traces of a student's submission and the reference solution. If one or more dissimilarities are found, the submission is decided as incorrect; otherwise, the submission is decided as correct. We have designed our method to find the inputs that will cause two programs to behave differently, by having either different output states or semantically different execution paths. Although it is simple to compare output states, proving the equivalence of execution paths is quite challenging. The goal of our work is to automatically decide the correctness of student programming assignments according to a reference implementation

ADVANTAGES:

- Reduces human efforts in writing test cases and makes the grading more complete

<u>SYSTEM REQUIREMENTS</u>

HARDWARE REQUIREMENTS:

Processor	-	intel i3 or i4
Speed	-	1.1 GHz
RAM	-	4 GB (min)
Hard Disk	-	500 GB(min)
Key Board	-	Standard Windows Keyboard
Mouse	-	Two or Three Button Mouse
Monitor	-	SVGA

SOFTWARE REQUIREMENTS:

Operating System	-	Windows 10 or above
Programming Language	-	Python