Using Static Analysis for Automated Assignment Grading in Introductory Programming Classes

Samuel Breese, Rensselaer Polytechnic Institute, breess@rpi.edu
Ana Milanova, Rensselaer Polytechnic Institute, milanova@cs.rpi.edu
Barbara Cutler, Rensselaer Polytechnic Institute, cutler@cs.rpi.edu

Abstract: Student experience in introductory computer science classes can be enhanced by applying static analysis techniques to automatically grade assignments. At Rensselaer Polytechnic Institute (RPI), introductory computer science classes (using Python) exceed 650 students in size. As resources are limited, it is infeasible to have teaching staff individually examine each student's answer for small in-lecture exercises; however, qualitative data regarding student code independent from execution is still valuable (and in some cases required) to assess progress. When static analysis utilities were made available to instructors and integrated with automatic assignment testing, instructors were able to judge student performance and provide feedback at a scale that would otherwise be infeasible.

There are clear advantages to applying static analysis techniques in comparison to less sophisticated methods (e.g. regular-expression based search). For one, students are unable to subvert grading by placing certain keywords within comments or string literals. Static analysis can also be applied to easily grade students on patterns that would be nontrivial to detect using a more naive method, for example in enforcing a rule that all member variables of a C++ class must be private, or verifying that a function takes the appropriate number and type of arguments.

Significance and Relevance of Topic: The introductory computer science course at RPI provides students with basic knowledge of programming concepts demonstrated through the Python programming language. Students are neither expected nor required to have prior programming experience. In such a foundational class, it is often more important that students demonstrate sufficient understanding of a concept than that they simply obtain the correct result. This can be done by administering multiple small exercises during each lecture. For example, early in the curriculum students are introduced to the "for" looping construct, and some time later they are introduced to the less-intuitive "while" looping construct. A simple problem might be given during lecture to give students hands-on experience with this new topic. Students would solve this problem, submit their solutions to Submitty[1], an open-source electronic grading platform developed at RPI, and receive a grade. By running each submission, Submitty can detect programs with the correct output, but previously there was no automated method by which to determine that students were actually using the new "while" loop: they might simply be solving the problem using the more intuitive "for" loop. Starting in the Fall 2016 semester, such functionality has been integrated with Submitty via the introduction of an open-source static analysis framework. Language support for this framework is now available to instructors in the majority of computer science courses offered at RPI.

These static analysis utilities were made available to instructors through a language-agnostic Python interface, so as not to require instructors teaching different courses to learn many different tools. This interface wraps lexical analysis and parsing tools for C, C++, Java, Python 2, and Python 3 developed in-house, giving an instructor access to token streams and abstract syntax trees for student code using familiar Python data structures. For particularly common use cases, scripts using this interface to count particular tokens, AST nodes, and function calls are provided: building upon the previous example, it is trivial for an instructor to distinguish programs using a "while"

[1] Submitty, Rensselear Center for Open Source Software, http://submitty.org

loop and programs using a “for” loop using the prebuilt script for counting AST nodes. Without such an integrated toolset, it becomes difficult to apply techniques used in one course to another due to the difference in tooling for different languages. The framework also serves to enforce a standard across multiple instructors using the same language: for example, given two C++ classes, one instructor might apply static analysis techniques using the LLVM compiler toolchain while another might use handwritten Lex/Yacc tools. Even if these two instructors were essentially performing the same analysis, their efforts would be duplicated and collaboration would be limited. This situation is avoided entirely by providing a consistent interface to all instructors.

Content: Automatic grading techniques using the framework described were applied to the Fall 2016 Computer Science 1 class at RPI, a class exceeding 650 students in size and involving 2 lecturers and 11 graduate teaching assistants. In the first 13 lectures, there were a total of 32 in-class exercises, graded exclusively by the automated grading system. In 18 of these exercises, instructors used static analysis techniques to enhance grading. Students received immediate feedback about the correctness of both the program output and adherence to other program requirements via static analysis. Using this feedback, students were able to correct any issues and make additional submissions. Static analysis techniques were also applied to each weekly homework assignment to supplement existing electronic grading and detailed manual TA grading and written feedback (typically comprising 25-50% of the total homework grade). To emphasize the degree to which this was utilized, there were an average of 7759 submissions to each homework assignment, and an average of 2472 submissions to each in-lecture assignment. It would clearly be infeasible for teaching staff to facilitate this level of student engagement entirely manually. Similar techniques were applied on a smaller scale to a C++ class, where analysis was used to prohibit the usage of certain language features (e.g., “goto” and “auto”).

Feedback from the instructors teaching the introductory course about the newly-available tools has been very positive. Teaching assistant workload for the mundane portions of homework grading (verifying adherence with specific assignment requirements) has been was significantly reduced, allowing the TAs to give more detailed and personalized feedback to each student regarding overall technique and style (e.g., program structure, good use of comments, and reasonable error checking) rather than focusing on programmatic minutia.

Adding support for the enhanced grading functionality to existing assignments has been eased by the aforementioned prebuilt scripts allowing an instructor to easily count the occurrences of a specific lexeme, AST node, or function call. In the majority of instances, instructors simply wanted to determine the existence of a specific number of applications of a given language feature, which amounted to no more than the addition of a single line in the automatic grading system assignment configuration file. Although some issues inevitably arose, they were relatively few, especially given the size of the class, the average volume of submissions, and the unfamiliarity of the students with the system. This was exacerbated by the class in consideration simultaneously transitioning from Python 2 to Python 3.

Our poster will contain examples of the instructor configurations required to grade a specific pattern in student code, showcasing the advantages over a regular-expression based approach by providing student code that will be correctly graded by static analysis but incorrectly graded by the textual approach. Additionally, as the end of the Fall 2016 semester approaches, more detailed numerical data relating to the number and type of analyses used and the number of students taking advantage of the enhanced feedback can be obtained and displayed. Furthermore, data relating to student performance on assignments in comparison to previous semesters (where static analysis was not available) will be provided in complete form.