Using Static Analysis for Automated Assignment Grading in Introductory Programming Classes

Samuel Breese, Ana Milanova, and Barbara Cutler

Rensselaer

Rensselaer Polytechnic Institute Computer Science

Submitty — Rensselaer Center for Open Source

Abstract

Student experience in introductory computer science classes can be enhanced by applying static analysis techniques to automatically grade assignments. Since resources for teaching large introductory programming courses 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.

A Motivating Use-Case

- Computer Science I at RPI is very large. Fall 2016 numbers: 650 students, 2 lecturers, and 11 graduate & ∼50 undergraduate TAs.
- Prior programming experience is not a prerequisite.
- Demonstrating knowledge of important concepts is often more important than producing the right final answer.
- Small exercises administered during lecture gauge student progress.
- Previously, our automated grading was purely output-based. Enforcement of special requirements was done manually by the TAs (or ignored).
- In Fall 2016, we introduced an easy-to-use static analysis framework.

Design Requirements

- Grading must be completely automatic.
- Instructor setup must be quick and simple.
- Tests should be robust to common student errors.
- Error messages to students should be clear, pointing out what went wrong and where to look for mistakes.
- Students should not be able to subvert grading requirements.

Examples of Instructor Usage

`for` vs. `while` loop in Python This simple code re-writing exercise is often used early in the semester, to help students learn the somewhat less intuitive `while` loop in Python.

```
# submission for assignment 1
for x in range(0,3):
    print(x)
```

(a) Initial code

```
# submission for assignment 1
x = 0
while x < 3:
    print(x)
    x += 1
```

(b) Student rewrite using `while`

```
"title" : "Loop Exercise",
"command" : [ "submitty_count_token for *.py",
              "submitty_count_token while *.py" ],
"points" : 1,
"validation" : [{
    "method" : "intComparison",
    "actual_file" : "STDOUT_0.txt",
    "description" : "Number of for loops",
    "comparison" : "eq",
    "term" : 0,
    "failure_message" : "Must not use a for loop",
},{
    "method" : "intComparison",
    "actual_file" : "STDOUT_1.txt",
    "description" : "Number of while loops",
    "comparison" : "ge",
    "term" : 1,
    "failure_message" : "Must use a while loop",
}]
```

(c) Typical Instructor Configuration

Restricting Use of `goto` in C++ Distinguishing the files below is not practical with a simpler approach (e.g., `grep`):

```
int main() {
x:
    goto x;
}
```

(d) Student code using goto

```
// goto
int main() {
    printf("goto");
}
```

(e) Student code without goto

Determining Maximum Loop Depth For more complex requirements, instructors can use the Python interface rather than using a provided script. The example below finds the depth of nested loops within student code.

```
for x in [1, 2, 3]:
    print(x)
for x in [1, 2, 3]:
    print(x)
```

(f) Loop Depth = 1

```
for x in [1, 2, 3]:
    for y in range(0, x):
        print(y)
    for z in [1, 2, 3]:
        print(z)
```

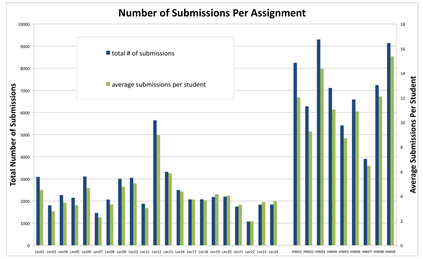
(g) Loop Depth = 2

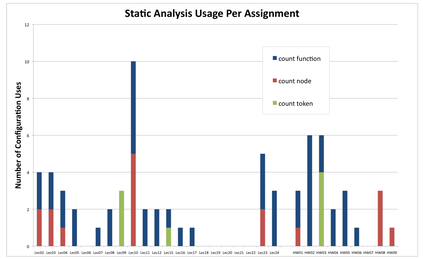
```
import lang.parser
paths = lang.parser.leaf_paths(
    lang.parser.python(open("student.py").read()))
print(max([len([x for x in path if x.name in ["for", "while"]])
           for path in paths]))
```

(h) Custom Grader to Determine Loop Depth

Results from Fall 2016

- These methods were introduced in the Fall 2016 introductory Computer Science I course at RPI.
- Students received immediate feedback about specification adherence.
- Students could correct any issues and make additional submissions.
- Infeasible for instructors to facilitate this level of student engagement.



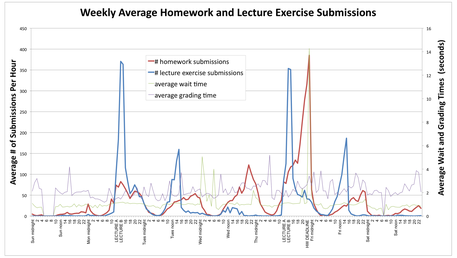


Details of Implementation

- Lexical analysis tool and parser available for C, C++, Python, and Java.
- Instructors use these tools through a language-agnostic Python interface.
- Scripts using this interface are provided for instructor convenience when dealing with particularly common use cases.
- Different tools on the backend are used for each language, but the instructor-visible interface is consistent, allowing cross-course sharing of configuration.

Performance

- At peak load (Thursday at midnight, the typical homework deadline for multiple courses), wait times for students averaged 15 seconds, despite thousands of submissions.
- Note the two spikes in usage on Monday and Thursday: these correspond to weekly lectures, demonstrating that students were actively submitting exercises and receiving feedback mid-lecture.
- Students are given a 24 hours to complete these exercises, explaining the smaller spikes on Tuesday and Friday.



Instructor Feedback

- Overall, feedback from instructors teaching the introductory course has been very positive.
- Teaching assistant workload for the mundane portions of homework grading (verifying adherence with specific assignment requirements) has been significantly reduced.
- Teaching assistants are free to give more detailed and personalized feedback to each student regarding overall technique and style rather than being forced to focus on programmatic minutia.

Ongoing Work

- Maintenance and bug fixing, security analysis.
- Increased language support.
- Expanded library of grading techniques for ease of use.
- Ensure more instructors are aware of the tool.

Submitty `http://submitty.org`

Submitty is an open source programming assignment submission system from the Rensselaer Center for Open Source Software (RCOS), launched by the Department of Computer Science at Rensselaer Polytechnic Institute.

Related Submitty Publications

- *User Experience and Feedback on the RPI Homework Submission Server*, Wong, Sihsobhon, Lindquist, Peveler, Cutler, Breese, Tran, Jung, and Shaw, SIGCSE 2016 Poster
- *A Flexible Late Day Policy Reduces Stress and Improves Learning* Tyler, Peveler, and Cutler, SIGCSE 2017 Poster
- *Submitty: An Open Source, Highly Configurable Platform for Grading of Programming Assignments*, Peveler, Tyler, Breese, Cutler, and Milanova, SIGCSE 2017 Demo Presentation

Acknowledgments

- Red Hat Software
- Rensselaer Center for Open Source (RCOS)
- CSCI 1100 Computer Science I Teaching Staff
- `https://github.com/Submitty/AnalysisTools`