Supporting Random Input and Automated Output Generation in Submitty

Drumil Patel [2] Evan Maicus [1] Matthew Peveler [1] Barbara Cutler [1]

[1] Rensselaer Polytechnic Institute [2] Indian Institue of Technology, Roorkee

Abstract

"Fuzzing," testing a codebase against a set of randomly generated inputs, has become a promising model of testing across the industry due to its ability to reveal difficult to detect bugs. Separately, the use of randomized inputs when testing student code submissions removes the potential for students to "hard-code" to known test cases. Motivated by these factors, we present a solution for the automated generation of testcase inputs and expected outputs within Submitty, an open source automated grading system from Rensselaer Polytechnic Institute. We detail an enhanced workflow that allows instructors to provide our testing system with an assignment-specific input generation script and an assignment solution. The input generation script is run at student test-time, providing students with either entirely generated inputs or a combination of generated and hand-crafted testcases. The instructor solution is run against the same inputs to produce expected results. This model of testcase specification carries the benefit of simple regeneration of expected output files if an assignment's specification changes or an inconsistency is identified.

Motivation

- Manual preparation of expected output files can be difficult, monotonous, and time consuming.
- Regeneration of expected output files due to changing assignment specifications can be costly.
- Randomly generated input file generation allows "fuzzing", revealing difficult to detect bugs.
- Static input across all students allow *hard coding* to known test cases.

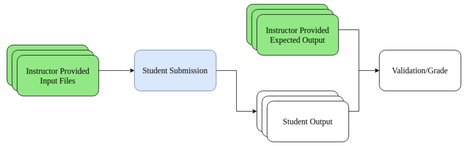
Acknowledgments

We are currently accepting applicants for Google Summer of Code 2020!

RCOS Rensselaer Center for Open Source

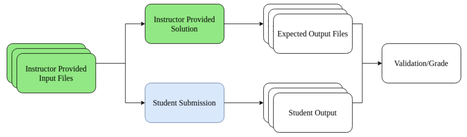
Google Summer of Code

Prior Submitty Workflow



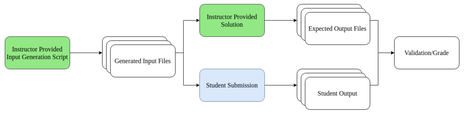
- The instructor creates an assignment specification.
- The Instructor hand crafts a number of testcase input against which to run student submissions.
- The Instructor either manually creates the expected output for each input— **or** — creates a working assignment solution and manually runs their solution against the test inputs to capture the resulting, expected output.
- The Instructor zips together a `Configuration JSON` file, their provided input files, and their expected output files, and uploads them to Submitty.

Generating Expected Output



- Instructor develops an assignment solution when configuring an assignment for use in Submitty.
  - This allows instructors to foresee potential common pitfalls students will encounter.
  - It also allows instructors to generate expected output files for use in autograding.
- Using this solution to generate or re-generate many output files can be tedious.
- Instructors to upload their solution and their input files, and their solution is used to automatically generate and re-generate expected output using the same infrastructure used to test and grade student submissions.

Input Generation

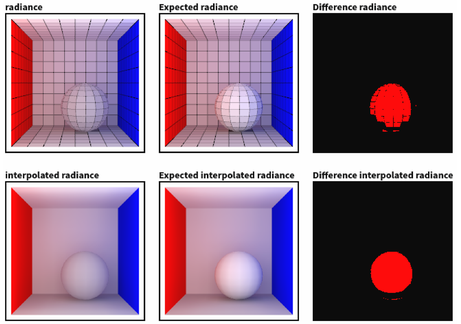


- The ability to generate expected output paves the way for automated input generation.
- With an instructor solution on-hand, expected output can be generated at the time a student submission is processed.
- We allow instructors to provide submitty with a *input generation script*.
- At test time, the input generation script generates inputs which are fed to both the instructor and student solution. The results are then compared.
- To allow per-student uniqueness, the input generation script is provided with the submitting student's id.
- To gain per-student per-submission uniqueness, the student's id and the current time can be used to seed a random number generator.
- Randomized testcases can be combined with hand crafted testcases to *fuzz* a student's application while guaranteeing that the important cases are covered.

Specification

```
{
  "title" : "Random Input Testcase",
  "input_generation_commands": [
    "python generator.py 1> rand.txt"
  ],
  "commands" : [
    "python submission.py rand.txt > student.txt"
  ],
  "solution_command" : [
    "python solution.py rand.txt > instructor.txt"
  ],
  "points" : 5,
  "validation" : [
    {
      "method" : "diff",
      "actual_file" : "student.txt",
      "expected_file" : "instructor.txt"
    }
  ]
}
```

Use in Advanced Topics Courses



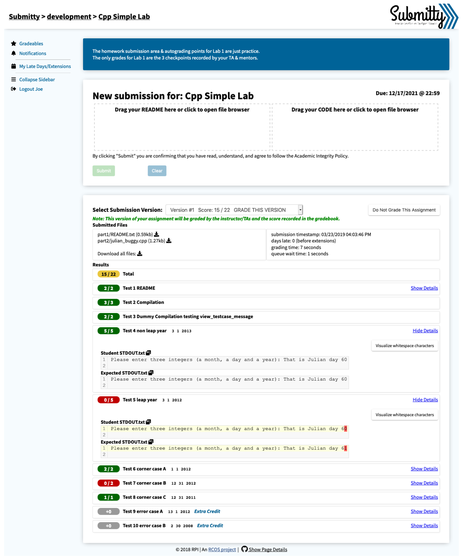
- Advanced topics assignments may involve output which is difficult to consistently and precisely capture.
- Assignments which are distributed across a network of containers produce distributed output which is tedious to manually prepare, but which can easily be gathered by Submitty.
- Graphical assignments produce non-text outputs in the form of screenshots and GIFs.
- To automate the grading of graphical assignments, programmatic keyboard, mouse, and standard input actions are taken against an assignment, which are impractical to consistently replicate by hand with pixel perfect precision.

Future Work

- Submitty is configured to use on average 10-20 testcases for an assignment. With the introduction of random input generation, *fuzzing* becomes possible, meaning that many hundreds of testcases can be performed against a given submission. The related user interface will be scaled accordingly as future work.
- Create infrastructure to allow inputs which are specified within an assignment configuration to be capture programmatically or from a live action demonstration (e.g. keyboard and mouse inputs).
- Allow input generation scripts which are written in compiled languages.

Submitty `https://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 Publications

- *Autograding Interactive Computer Graphics Assignments*
  Evan Maicus, Matthew Peveler, Andrew Aikens, Barbara Cutler
  SIGCSE 2020 Proceedings
- *Autograding Distributed Algorithms in Networked Containers*
  Evan Maicus, Matthew Peveler, Stacy Patterson, and Barbara Cutler
  SIGCSE 2019 Proceedings
- *Comparing Jailed Sandboxes vs Containers Within an Autograding System*
  Matthew Peveler, Evan Maicus, and Barbara Cutler
  SIGCSE 2019 Proceedings