Patch based test coverage for quick test feedback

Google Summer of Code 2023 Proposal.

Takemaru Kadoi(diohabara@gmail.com)

Overview

- **Project size: medium**
- **Difficulty: Simple**
- **Confirmed Mentors: @hnrklssn**

The LLVM regression tests are executed by Lit[1], which is structured as source code or IR to be passed to some binary, rather than test code directly calling the code to be tested. This has many advantages but can make it difficult to predict which code path is executed when the compiler is invoked with a certain test input, especially for edge cases where error handling is involved. The goal of this project is to help developers create good test coverage for their patch and enable reviewers to verify that they have done so. To accomplish this, a tool can be fed a patch as input, add coverage instrumentation for the affected source files, runs Lit tests, and records which test cases cause each counter to be executed. For each counter, we can then report the number of test cases executing the counter, but perhaps more importantly we can also report the number of test cases executing the counter that are also changed in some way by the patch, since a modified line that results in the same test results isn't properly tested unless it's intended to be a non-functional change.

Implementation

The proposed project will be implemented in three parts:

1. **Adding option to llvm-lit**
   - Add an option to llvm-lit to emit the necessary test coverage data, divided per test case.
   - This will involve setting a unique value to LLVM_PROFILE_FILE for each RUN.
2. **New tool for processing coverage data and git patch**
   - Create a new tool to process the generated coverage data and the relevant git patch.
   - The tool should present the results in a user-friendly manner.

[1]https://www.llvm.org/docs/TestingGuide.html

    - Adding non-intrusive way to enable coverage instrumentation
3. **Add a way to non-intrusively (without changing build configurations) enable coverage instrumentation to a build.**
    - By building the project normally, touching the files changed by the patch, and rebuilding with CCC_OVERRIDE_OPTIONS set to add coverage, we can lower the overhead of generating and processing coverage of lines not relevant to the patch.
    - The tooling in step 2 and 3 can be made completely agnostic of the actual test-runner, making it easier for other test harnesses besides Lit to implement the same functionality. If time permits, adding this as a step in CI would also be helpful for reviewers.

Expected results

- **Help developers find uncovered edge cases**: The tool will aid developers in identifying uncovered edge cases during development. This will allow them to create better test coverage for their patches.
- **Avoid adding unnecessary code**: The tool will also help developers avoid adding unnecessary asserts or logs just to check that the code is actually executed.
- **Assist reviewers in verifying patch tests**: Reviewers will have an easier time verifying which parts of the patch are tested by each test using the tool.

Objectives

- Introduce a tool to help LLVM developers create good test coverage for their patches.
- Enable reviewers to verify that proper test coverage has been implemented.
- Add coverage instrumentation for affected source files.
- Run Lit tests and record which test cases cause each counter to be executed.
- Make the tool agnostic of the actual test-runner, lowering the threshold for other test harnesses to implement the same functionality.
- Enable the tool to be used by the community to help developers find uncovered edge cases during development and enable reviewers to check which parts of the patch are tested by each test.

Roadmap

Week 1

- Get familiar with the LLVM testing infrastructure and code coverage tools.

Week 2-3

- Implement the first part of the project - adding an option to llvm-lit to emit necessary test coverage data.

- Write tests to ensure proper functioning of the added option.

Week 4-5

- Implement the second part of the project - creating a new tool to process generated coverage data and the relevant git patch.
- Write tests to ensure proper functioning of the new tool.

Week 6-7

- Implement the third part of the project - adding a way to non-intrusively enable coverage instrumentation to a build.
- Write tests to ensure proper functioning of this feature.

Week 8-9

- Integrate the three parts of the project and make the tool agnostic of the actual test-runner.
- Test the integrated tool with various LLVM tests.

Week 10-11

- Refine the tool based on feedback from mentor and community.
- Add documentation and examples for users to help them get started with the tool.

Week 12

- Set up a CI pipeline to automate the testing and reporting of coverage data.
- Write tests to ensure the proper functioning of the CI pipeline.

Week 13

- Finalize the project and prepare it for submission.
- Write a report documenting the project and its outcomes.

Deliverables

- Option to emit necessary test coverage data in llvm-lit.
- New tool to process generated coverage data and relevant git patch.
- Way to non-intrusively enable coverage instrumentation to a build.
- Integrated tool that is agnostic of the actual test-runner.
- Documentation and examples for users.
- CI pipeline for automated testing and reporting of coverage data.
- Report documenting the project and its outcomes.

About Me

Personal Information

- Name: Takemaru Kadoi
- Email: diohabara@gmail.com
- LinkedIn: https://www.linkedin.com/in/takemaru-kadoi/
- GitHub: diohabara
- Discourse on LLVM: diohabara
- Discord on LLVM: diohabara
- Residensy: CST(UTC - 06:00)
- Availability over the project
  - Up to 40 hours per week during `2023/05/13-2023/08/20`
  - No courses, no internships
- Operating systems and hardware
  - Linux(amd64), macOS(AArch64)

Why Me?

As a PhD student with a focus on programming language and experience contributing to several open source projects[^gccrs1][2], I have developed a deep understanding of programming languages and their intricacies. My passion for testing has led me to explore various testing methodologies and tools, including LLVM's Lit testing framework.

In addition, my experience as an officer in the computer security group at my university[^csg][3] has taught me the importance of thoroughly testing code to ensure its security and reliability. Through my internships[4] at several companies, I have honed my skills in software development and collaboration.

Overall, my combination of technical expertise, testing knowledge, and experience working with open source projects make me an ideal candidate for the "[Coverage] Patch based test coverage for quick test feedback" project.

[2] Commit in C++ https://github.com/Rust-GCC/gccrs/commit/66832f312a7db436110a3b08ff51eac349d5fdbf

[3] Write-ups for CTF challenge hosted by NSA. https://github.com/diohabara/nsa-codebreaker-challenge2022

[4] I have worked at several companies as intern and learned how to write technical documents.