Project 2 Testing report

cs61bl-ba
cs61bl-am
cs61bl-an

Introduction

We organized our testing in three main groups, the same groups we developed this project in: Expression and expression tree (ProofTree), Parsers and Commands.

Each testing group has a set of characteristics to ensure we check every possible error.

Expression and ProofTree tests:

Expressions:

We tested all the possible expression creations:

- all sorts of legal constructions
    - no-operator expressions
    - single operator expressions
    - multiple operator expressions
- all sorts of illegal constructions
    - no parenthesis
    - empty expressions
    - illegal characters expressions
    - incomplete parenthesis or incorrectly parenthesized expressions.

(and many more).

If we had bugs on the creation of expressions they would easily show up on our tests as they are very exhaustive.

ProofTree:

As ProofTree is the representation of the expression in a tree form, we had to account for the three comparison operators we provided: equal, equalOppositeSign and isEquivalent.

We didn't test the iterators, as they were identical to code submitted during labs this semester and were correct.

All three tests provided tests cases for different kinds of non-trivial trees. We checked cases in which the tree should be equal/equivalent and cases in which it should not.

In the isEquivalent test (required for theorem use) we made sure that we added special cases. We tested definition expressions that shared subexpressions on both sides of operators, for example: theorem ((a&b)=>a).

If we had bugs on comparison (a key element in our project) we would rapidly see them reflected on this tests and it would save a lot of time.

Parsers and Proof:

Parsers:

The parsers are a key part of the project and as such we set up exhaustive tests. They check for **both incorrect formatting** and **most of the scoping errors (errors involving accessing lines out of our reach)**.

We set up two test methods that albeit simple, account for incorrect formatting on all commands and scoping access.

We test this by adding extra arguments or no arguments on all our commands, and different illegal scope access commands.

If there was a bug in the parser, allowing a line to be incorrectly formatted or an illegal scope access, it would show up in the tests.

Proof:

This is the definite test, a lot of commands are tested in combination and if our program has a bug chances are it will show up here.

In this test we executed proofs that would finish with a single proof and multiple subproofs, thus covering the entire proving spectrum.

Commands:

We tested all the non trivial command classes that are correctly formatted for a correct execution (as the command creation is already handled by our parser).

We excluded the DummyCommand, ShowCommand classes as they virtually do nothing complicated. The print command is hardcoded in our program by the use of the DummyCommand.

In every execution we would test for two things:

- the correct exception throwing in case the syntax was incorrect.
    - we didn't just test a general exception but the correct kind of exception
- the correct inference assignation, in case the command was executed properly.

Every command has it's own characteristics but they can be separated in two groups:

- commands that can have the arguments switched
    - we tested the same inputs twice, switching the arguments
- commands that have one argument

Each command has one or two cases in which it should work and one case in which it should fail.

Contributions:

The repository of this project (along with all the commit history) is located here: https://github.com/mllobet/proof-check , this serves as a log of how we have worked throughout the project and each member's contribution. (Graphs and stats located here: https://github.com/mllobet/proof-check/graphs)

ba (mllobet):
Contributions: Commands, Proof, ProofTree.
Implemented and tested all the Commands and their integration with the proof. Worked on the Tree equivalence and iterators as well.

am (Kevin Devroede):
Contributions: Parsing, parsing, parsing!
Parsing was a big part of our project and required the exclusive dedication of a member. am spent the majority of the time making everything work in this aspect.

an (ccirclaeys):
Contributions: ProofTree and Expressions.
Developed the Tree and Expressions early on in the project to provide the team with a usable data structure.