Advanced Programming 2024

Assignment Five

Property-Based Testing

Due: 2024-10-13

Synopsis: Testing of functionality from previous assignments using QuickCheck

1 Introduction

In this assignment you will do property-based testing of components from earlier assignments: printer, parser, evaluator and type checker. Specifically, you will be implementing the following:

- A good quality generator for expressions.
- A property for the interaction of parsing and printing.
- A property for the interaction of type checking and evaluation.
- Bug fixes revealed by the properties.

2 Task: A better generator

In this task you will modify the expression generator to improve the distribution of generated expressions.

Change the definition of `genExp` so that the `expCoverage` property passes. In particular this means that

- Between 20% and 80% of expressions should have domain errors (division by zero or negative exponent) reported by `checkExp`.
- Between 20% and 80% of expressions should have type errors reported by `checkExp`.
- Between 5% and 30% of expressions should have variable errors reported by `checkExp`.

- At least 50% of expressions should contain a variable between 2 and 4 characters long.

You *are* permitted to change the signature of `checkExp`, but make sure to also change the definition of `arbitrary` for `Exp` accordingly.

2.1 Suggested Implementation

Use `frequency` instead of `oneof` to better control the distribution. Keep track of the variables in scope by adding an additional parameter of type `[VName]` to `genExp`.

3 Task: A property for parsing/printing

In this task you will implement the property `parsePrinted`, which checks that printing an expression and parsing the resulting string gives back the original expression.

When QuickCheck finds a counter-example to `parsePrinted` you must decide if the error is in the *implementation*, the *generator* or the *property*. Then fix the error and continue to test until QuickCheck passes the property with at least 10000 tests (see the course notes for how to change the number of tests).

4 Task: A property for checking/evaluating

In this task you will implement the property `onlyCheckedErrors`, which checks that evaluating an expression only results in those errors detected by `checkExp`.

Note that `checkExp` now returns a *list* of errors. The intention is that this list contains all errors that *might* occur during evaluation.

Define `onlyCheckedErrors` so that it evaluates the given expression; if the evaluation returns an error, that error must be in the list returned by `checkExp`. Check the property with as many tests as necessary to produce a counterexample.

For this task you do not need to fix the implementation, but you should analyse and describe the problem (see report requirements).

5 Code handout

The code handout consists of the following nontrivial files.

- `a5.cabal`: Cabal build file. **Do not modify this file.**
- `runtests.hs`: Test execution program. **Do not modify this file.**
- `src/APL/AST.hs`: The APL AST definition. **You may only modify `printExp` in this file.**
- `src/APL/Error.hs`: The APL Error definition. **Do not modify this file.**
- `src/APL/Parser.hs`: The parser.
- `src/APL/Check.hs`: The type checker.
- `src/APL/Eval.hs`: The APL evaluation function.
- `src/APL/Tests.hs`: The property tests.

6 Your Report

You are expected to comment on the *interesting* details of your implementation. You are *not* expected to give a line-by-line walkthrough of your code. Most importantly, you are expected to reflect on the *quality* of your code:

- Do you think it is functionally correct? Why or why not?
- Is there some improvement you'd have liked to make, but didn't have the time?

It is more important to be aware of the strengths or shortcomings of your solution, than it is to have a complete solution.

6.1 The structure of your report

Your report must be structured exactly as follows:

Introduction: Briefly mention very general concerns, your own estimation of the quality of your solution, and possibly how to run your tests.

A section for each task: Mention whether your solution is functional, which cases it fails for, and what you think might be wrong.

A section answering the following numbered questions:

1. Can programs produced by your generator loop infinitely? If so, would it be possible to avoid this?
2. What counter-examples did `parsePrinted` produce? For each counter-example, which component (implementation, generator or property) did you fix?
3. What is the mistaken assumption in `checkExp`?

All else being equal, **a short report is a good report**.

7 Deliverables for This Assignment

You must submit the following items:

- A single PDF file, A4 size, no more than 5 pages, describing each item from report section above.
- A single zip/tar.gz file with all code relevant to the implementation, including at least all the files from the handout. For this assignment it is not necessary to add additional files.

Remember to follow the general assignment rules listed on the course homepage.

8 Assessment

You will get written qualitative feedback, and points from zero to four. There are no resubmissions, so please hand in what you managed to develop, even if you have not solved the assignment completely.