Assignment 5

Advance Programming

Name: Kazi Sohrab Uddin Titu, Group Number: 58

1 Introduction:

This assignment consists of some features like to enhance the distribution of generated expressions and ensure that they meet certain properties through QuickCheck testing. I estimate the quality of my solution to be quite satisfactory, as the core functionalities work as expected; however, some edge cases and error handling scenarios may not have been fully addressed. I have tried to ensure coverage of different cases. Though there are some edge cases that could be refined further but I tried to cover them up. To run test cases, run cabal test in the terminal.

2 Task: A Better Generator

Solution Functionality:
The objective of this task was to modify the genExp function to improve the distribution of generated expressions. My solution implements a frequency-based approach to control the likelihood of various expression types, which helps in achieving the desired distribution of errors. I used genVar to generate variable names with a mix of letters and digits, controlling their length, genPos to produce positive integers to ensure valid operations, especially in divisions and genLam to create lambda expressions, incorporating new variable names into the expression generation. The current implementation passes the general requirements, but it fails to meet the specific coverage criteria:

- **Domain Errors**: The implementation produces an adequate proportion of domain errors (division by zero or negative exponent) but needs to be adjusted to ensure it falls between 20% and 80%.
- **Type Errors**: Similar to domain errors, type errors are present but require adjustment to meet the specified range.
- **Variable Errors**: The frequency of variable errors is lower than desired, necessitating an increase to ensure it lies within the 5% to 30% range.

- **Non-Trivial Variables**: The implementation ensures that at least 50% of expressions contain variables between 2 and 4 characters long.

Potential Issues:
I believe the incorrect distributions may stem from the current frequency weights used in genExp. Adjusting these weights should allow for better coverage of the error types.

3 Task: A Property for Parsing/Printing

Solution Functionality:
The initial implementation successfully checks the equivalence of the original expression and the parsed expression. However, QuickCheck encountered counterexamples indicating potential issues with the parsePrinted function.

Failures or Limitations:
The parsePrinted property failed consistently for various test cases, including expressions such as CstBool False and CstInt 1. These failures indicate that the conversion from the expression to its string representation using show does not produce a format that the parseAPL function can successfully parse back into the original expression. The root cause may lie in the show implementation or in how parseAPL interprets the string representation, leading to a mismatch between the original and parsed expressions.

Due to the consistent nature of the failures, further investigation is needed to determine the exact issue in either the string representation or the parsing logic to ensure the correctness of the parsePrinted property.

4 Task: A Property for Checking/Evaluating

Solution Functionality:
The onlyCheckedErrors function evaluates the given expression using the eval function and captures any errors returned. It then compares these errors to the list of errors reported by

checkExp. If evaluation returns an error, the function checks whether this error is included in the list of checked errors. If no errors occur during evaluation, the function returns True.

Potential Issues:
While the onlyCheckedErrors property is designed to ensure that all errors encountered during evaluation are accounted for by checkExp, it may not work as intended in all cases. Potential issues include:

1. **Incomplete Error Detection**: If checkExp does not cover all possible errors that could arise during evaluation, this property could pass incorrectly, indicating that all errors are checked when they are not.
2. **Error Representation**: If the error representation in checkExp differs from that produced by eval, the comparison may fail, leading to false positives where the evaluation errors are not properly accounted for.
3. **No Guarantee of Exhaustive Checks**: The property does not guarantee that all errors are checked—only those that were actually encountered during the evaluation of a specific expression. This means that some expressions may pass the test even if they contain potential errors not captured by checkExp.

5 Questions:

1. Can programs produced by your generator loop infinitely? If so, would it be possible to avoid this?
Yes, programs generated by the genExp function can potentially loop infinitely, especially in the presence of recursive constructs (e.g., Let, Lambda, or TryCatch expressions that reference themselves). To avoid infinite loops, constraints could be introduced to limit recursion depth or ensure that recursive calls have a base case that eventually terminates.

2. What counter-examples did parsePrinted produce? For each counter-example, which component (implementation, generator, or property) did you fix?
The parsePrinted property produced counter-examples for cases like CstBool False and CstInt 1. This failure indicates an issue with the implementation of the parsePrinted function, which does not accurately match the generated expressions against their printed representations. Fixing the implementation would involve ensuring that the parsing and evaluation of expressions are consistent and that all expression forms are properly handled during the printing process.

3. What is the mistaken assumption in checkExp?
A mistaken assumption in checkExp is that it can accurately determine all potential errors that may arise during expression evaluation. If checkExp fails to account for specific edge cases or types of errors, the evaluation process might produce errors that are not listed by checkExp. This gap can lead to misleading results in properties that depend on the correctness of checkExp, such as onlyCheckedErrors.