САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление: 02.03.01 Математика и компьютерные науки

Отчет по практическому заданию №5

по дисциплине: «Функциональное программирование»

Студент:
группы 5130201/20102
_____ Салимли А.

Преподаватель:
к.т.н.
_____ Моторин Д.Е.

«____»_____2024 г.

Санкт-Петербург, 2024 г.

Содержание

Введение

В данном отчете, описана реализация и результаты тестирования QuickCheck. Тестирование реализовано для двух заданных математических функций, на проверку их свойств.

Все тесты были записаны в Spec.hs.

Для работы с QuickCheck, необходимо прописать в dependencies в package.yaml или в .cabal: следующее:

- QuickCheck

Ограничения:

Функции нужно записать в библиотеку Lib.hs и ограничить доступ к вспомогательным функциям.

1. Постановка задачи

Для реализации 5-ой лабораторной работы, по дисциплине «функциональное программирование», необходимо создать проект в stack. В проекте stack, есть директория tests, в которой есть файл Spec.hs, где можно так же прописать main, для вывода результатов тестирования.

Необходимо проверить две функции:

(1) Функция умножения по модулю multiplyMod

- Тип: Int -> Int -> Int -> Int
- Принимает три целых числа: два множителя и модуль и возвращает произведение двух чисел по модулю.

Проверить:

1. Умножение по модулю: (multiplyMod x y m) mod m == (x * y)mod m
2. Нейтральный элемент: multiplyMod x 1 m == x mod m
3. Коммутативность: multiplyMod x y m == multiplyMod y x m

(2) Функция GCD

- Тип: Int -> Int -> Int
- Вычисляют наибольший общий делитель (НОД) двух целых чисел.

Проверить:

1. GCD числа самим собой равен самому числу.
2. GCD числа с 1 равен 1.
3. GCD коммутативен (gcd a b == gcd b a).
4. GCD ассоциативен (gcd (gcd a b) c == gcd a (gcd b c)).

Так как в стандартной библиотеки Haskell : Prelude, уже есть функция gcd, может возникнуть конфликт имен. Следовательно нужно либо:

1. Прописать в начале import Prelude hiding (gcd)
2. Переименовать функцию на myGcd (выберем этот вариант)

2. Реализация функций

2.1 Функция multiplyMod

Функция реализована в Lib.hs:

```
multiplyMod :: Int -> Int -> Int -> Int
multiplyMod x y m = (x * y) `mod` m
```

Функция вычисляет результат умножения двух целых чисел с последующим взятием остатка от деления на модуль m.

Тут x - первое число в умножении, y - второе число, m - модуль, на который вычисляется остаток.

Пример:

multiplyMod 5 7 3 = 5 * 7 mod 3 = 35 mod 3 = 2

2.2 Функция GCD

Функция вычисляет наибольший общий делитель (НОД) двух целых чисел a a и b b с использованием алгоритма Евклида.

```
myGcd :: Int -> Int -> Int
myGcd a b
  | b == 0 = abs a
  | otherwise = myGcd b (a `mod` b)
```

Тут сначала рассматриваются базовый и рекурсивный случаи:

Если b = 0, то результат и есть первое число = | a |

Если b != 0, то вызывает себя с параметрами: b: делитель, a mod b: остаток от деления.

Таким образом, функция постепенно уменьшает значения a и b, пока b не станет = 0.

Пример:

myGcd 9 6 = (1) 9 mod 6 = 3 => (2) 6 mod 3 = 0 => (3 базовый случ.) 3 mod 0 = 3.

=> myGcd 9 6 = НОД (9, 6) = | 3 | = 3.

3. Тестирование

В тесте необходимо проверить свойства описаные в пункте 1, соответственно.

3.1 Тестирование для multiplyMod

prob_multiplyMod_mod: Проверяет, что результат умножения по модулю совпадает с прямым вычислением остатка.

$$(x \times y) \mod m = \text{multiplyMod}(x, y, m) \mod m$$

prop_multiplyMod_identity: Проверяет, что если один из множителей = 1 =>
результатсовподает с x mod m.

$$\text{multiplyMod}(x,1,m) = x \mod m$$

prop_multiplyMod_commutative: Проверяет коммутативность умножения по модулю. То есть, что порядок множителей не влияет на результат.

$$\text{multiplyMod}(x, y, m) = \text{multiplyMod}(y, x, m)$$

3.2 Тестирование для GCD

prop_myGcd_self: Проверяет, что НОД любого числа с самим собой равен абсолютному значению числа.

$$\text{myGcd}(x, x) = |x|$$

prop_myGcd_one: Проверяет, что НОД любого числа x с 1 = 1.

$$\text{myGcd}(x,1) = 1$$

prop_myGcd_commutative: Проверяет, что НОД коммутативен, то есть порядок аргументов не влияет на результат.

$$\text{myGcd}(a, b) = \text{myGcd}(b, a)$$

prop_myGcd_associative: Проверяет ассоциативность НОД, то есть результат зависит только от значений чисел, но не от порядка группировки

$$\text{myGcd}(\text{gcd}(a, b), c) = \text{myGcd}(a, \text{myGcd}(b, c))$$

Spec.hs (main):

Выводит заголовки для каждой группы тестов (для функций multiplyMod и myGcd).

Вызывает QuickCheck для каждого свойства, чтобы проверить его на большом наборе случайных данных.

Если тесты проходят, QuickCheck сообщает об успешных проверках. Если находятся ошибки, QuickCheck выведет пример, на котором свойство нарушается.

Исходный код:

```haskell
module Main (main) where
import Test.QuickCheck
import Lib (multiplyMod, myGcd)

prop_multiplyMod_mod :: Int -> Int -> Int -> Property
prop_multiplyMod_mod x y m = m > 0 ==> (multiplyMod x y m) `mod` m == (x * y) `mod` m

prop_multiplyMod_identity :: Int -> Int -> Property
prop_multiplyMod_identity x m = m > 0 ==> multiplyMod x 1 m == x `mod` m

prop_multiplyMod_commutative :: Int -> Int -> Int -> Property
prop_multiplyMod_commutative x y m = m > 0 ==> multiplyMod x y m == multiplyMod y x m

prop_gcd_self :: Int -> Bool
prop_gcd_self x = myGcd x x == abs x

prop_gcd_one :: Int -> Bool
prop_gcd_one x = myGcd x 1 == 1

prop_gcd_commutative :: Int -> Int -> Bool
prop_gcd_commutative a b = myGcd a b == myGcd b a

prop_gcd_associative :: Int -> Int -> Int -> Bool
prop_gcd_associative a b c = myGcd (myGcd a b) c == myGcd a (myGcd b c)

main :: IO ()
main = do
  putStrLn "Тестики для умножения по модулю:"
  quickCheck prop_multiplyMod_mod
  quickCheck prop_multiplyMod_identity
  quickCheck prop_multiplyMod_commutative
  putStrLn "Тестики для НОД:"
  quickCheck prop_myGcd_self
  quickCheck prop_myGcd_one
  quickCheck prop_myGcd_commutative
  quickCheck prop_myGcd_associative
```

Результаты тестирования

Для тестирования функций необходимо в директории папки с проектом прописать команду:

```
stack test
```

После чего увидем результаты тестов прописаных в Spec.hs:

Тестики для умножения по модулю:

+++ OK, passed 100 tests; 97 discarded.

+++ OK, passed 100 tests; 123 discarded.

+++ OK, passed 100 tests; 108 discarded.

Тестики для НОД:

+++ OK, passed 100 tests.

+++ OK, passed 100 tests.

Результаты тестирования

Тут N discarded - означает, что QuickCheck сгенерировал N случайных наборов данных, которые не соответствовали условиям теста, например, потому что m <= 0 для тестов с модулем. QuickCheck продолжает генерировать данные, пока не наберёт достаточно случаев, которые проходят проверку условия.

Main.hs:

```haskell
module Main where
import Lib (multiplyMod, myGcd)

main :: IO ()
main = do
  putStrLn "Умнож. по модулю:"
  putStrLn $ "3 4 5 = " ++ show (multiplyMod 3 4 5) -- 2
  putStrLn $ "10 15 7 = " ++ show (multiplyMod 10 15 7)
  putStrLn $ "20 30 6 = " ++ show (multiplyMod 20 30 6)
  putStrLn "\nНОД:"
  putStrLn $ "30 45 = " ++ show (myGcd 30 45)
  putStrLn $ "17 23 = " ++ show (myGcd 17 23) -- 1
  putStrLn $ "100 25 = " ++ show (myGcd 100 25)
```

В Main.hs, привожу пример работы программы вычисления multiplyMod x y m и GCD x y.

Проверим результат:

multiplyMod 3 4 5 <=> (3 x 4) mod 5 = 12 mod 5 = 2 : Верно

GCD 17 23 <=> 23 mod 17 => 17 mod 6 => 6 mod 5 => 5 mod 1 = 0 => 1 : Венро

4. Заключение

В результате выполнения лабораторной работы №5, были изучены основы тестирования ПО, на языке Haskell, с использованием библиотеки QuickChek написанный в stack проекте в Spec.hs.

Были протестированы математические функции на их свойства:

(1) Умножение по модулю multiplyMod

(2) Наибольший общий делитель GCD

В ходе тестирования, обе функции «прошли» все тесты. Что подтверждает корректность свойств функций в нашей программе.

5. Список источников

[1] W. Kurt. Get Programming with Haskell. Москва: ДМК, 2019. 481 с.

[2] В. Брагилевский. Haskell in Depth. Москва: ДМК, 2019. 340 с.

6. Приложение А

Ссылка на репозиторий с кодом: https://github.com/MathematicLove/Haskellove