МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ 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 №4

по дисциплине «Функциональное программирование»

Обучающийся: _____ Гладков И.А.

Руководитель: _____ Моторин Д.Е.

«_____»_____20____г.

Санкт-Петербург, 2024

Содержание

Введение

В данном практическом задании необходимо выполнить следующие задачи:

1. Написать функцию логарифма `logBase :: Double -> Double -> Double`, которая принимает два числа: основание и аргумент, и возвращает логарифм аргумента по заданному основанию. Используя `QuickCheck`, проверьте следующие свойства:
   (a) Обратное свойство: `logBase b (b ** x) == x` для любого положительного `b` и `x`.
   (b) Свойство смены основания: `logBase a b == logBase c b / logBase c a` для любых положительных `a, b и c`, где `c` не равно 1.
   (c) Логарифм единицы: `logBase b 1 == 0` для любого положительного `b`.
2. Написать функцию matrixMultiply :: Num a => [[a]] -> [[a]] -> [[a]], которая выполняет умножение двух матриц. Используя QuickCheck, проверьте следующие свойства:
   (a) Ассоциативность: `(A *B) * C == A * (B * C)` для любых матриц `A, B и C` (при условии, что размеры матриц позволяют это умножение).
   (b) Дистрибутивность:`A * (B + C) == A*B + A * C` для любых матриц `A, B и C`.
   (c) Умножение на единичную матрицу: `A * 1 == A`, где `I` — единичная матрица соответствующего размера.

Лабораторная работа выполнена на языке Haskell в текстовом редакторе Visual Studio Code 1.95.3.

1 Математическое описание

QuickCheck — это инструмент для автоматической проверки свойств программ. Он использует метод генерации тестов с последующей проверкой утверждений о программе с использованием случайных данных. QuickCheck позволяет описывать свойства функций в виде логических выражений, и затем автоматически проверять их с помощью случайных значений.

В Haskell для работы с QuickCheck есть несколько основных команд и функций, которые позволяют генерировать тесты и проверять свойства:

- **quickCheck:** Команда для запуска проверки свойства. Она принимает функцию с описанием свойства и автоматически генерирует случайные входные данные для тестирования.
- **Property:** Это тип, который представляет собой проверку логического свойства, которое может быть либо истинным, либо ложным, в зависимости от входных данных. Он часто используется с операторами, такими как ==>, чтобы описать свойства, которые должны выполняться при определённых условиях.

Для того чтобы QuickCheck мог генерировать случайные данные для тестов, необходимо определить, как создавать случайные значения для тех типов данных, которые используются в тестируемых функциях. Это делается через класс Arbitrary.

Arbitrary — это класс типов, который предоставляет метод arbitrary, используемый для генерации случайных значений. Для каждого типа, для которого мы хотим генерировать случайные значения, нужно предоставить инстанс этого класса.

2 Особенности реализации

2.1 Функция `logBase`

`logBase` – функция, принимающая два числа: основание и аргумент, и возвращающая логарифм аргумента по заданному основанию. Функция представлена в листинге 1.

```haskell
logBase :: Double -> Double -> Double
logBase a b = log b / log a
```

Листинг 1. Функция `logBase`

Для проверки равенства значений с плавающей точкой была создана функция `almostEqual`, проверяющая их примерное равенство с заданной точностью.

```haskell
almostEqual :: Double -> Double -> Double -> Bool
almostEqual a b epsilon = abs (a - b) < epsilon
```

Далее были написаны тесты для данной функции в файле `Spec.hs`:

1. Обратное свойство: `logBase b (b ** x) == x` для любого положительного `b` и `x`.

```haskell
    prop_reverse :: Double -> Double -> Property
    prop_reverse b x = b > 0 && x > 0 && b /= 1 ==> almostEqual first second 1e-9
    where
    first = logBase b (b ** x)
    second = x

```

2. Свойство смены основания: `logBase a b == logBase c b / logBase c a` для любых положительных `a, b и c`, где `c` не равно 1.

```haskell
    prop_change :: Double -> Double -> Double -> Property
    prop_change a b c = a > 0 && b > 0 && c > 0 && c /= 1 && a /= 1 ==> almostEqual
    ↪first second 1e-9
    where
    first = logBase a b
    second = logBase c b / logBase c a

```

3. Логарифм единицы: `logBase b 1 == 0` для любого положительного `b`.

```haskell
    prop_one :: Double -> Property
    prop_one b = b > 0 && b /= 1 ==> almostEqual first second 1e-9
    where
    first = logBase b 1
    second = 0

```

2.2 Функция matrixMultiply

matrixMultiply – функция, выполняющая умножение двух матриц. Код функции представлена в листинге 2.

```
matrixMultiply :: Num a => [[a]] -> [[a]] -> [[a]]
matrixMultiply a b
| colsA /= rowsB = error "Incompatible matrix dimensions"
| otherwise = map (\row -> map (sum . zipWith(*) row) (transpose b)) a
where
colsA = length (head a)
rowsB = length b
```

Листинг 2. Функция matrixMultiply

Она использует функция транспонирующую матрицу – transpose

```
transpose :: [[a]] -> [[a]]
transpose ([]:_) = []
transpose x = map head x : transpose (map tail x)
```

Была написана функция создающая матрицу заданных размеров – genMatrix

```
genMatrix :: (Arbitrary a) => Int -> Int -> Gen [[a]]
genMatrix rows cols = replicateM rows (vectorOf cols arbitrary)
```

Для проверки матриц, содержащих значения с плавающей точкой, используется функция проверяющая матрицы на примерное соответствие с заданной точностью.

```
almostEqualMatrices :: [[Double]] -> [[Double]] -> Double -> Bool
almostEqualMatrices left right epsilon =
if null left || null right then
left == right
else
let matrixLeft = concat left
matrixRight = concat right
in length matrixLeft == length matrixRight &&
all (\(l, r) -> almostEqual l r epsilon) (zip matrixLeft matrixRight)
```

Далее были написаны тесты для данной функции в файле Spec.hs:

1. Ассоциативность: (A *B) * C == A * (B * C) для любых матриц A, B и C (при условии, что размеры матриц позволяют это умножение).

   Для генерации корректных матриц, подходящих под условия проверки условия написана функция genAssociativeMatrices

```
    genAssociativeMatrices :: (Arbitrary a) => Gen ([[a]], [[a]], [[a]])
    genAssociativeMatrices = do
    m <- chooseInt (1, 5)  -- строки в матрице A
    n <- chooseInt (1, 5)  -- столбцы в A и строки в B
```

```
    p <- chooseInt (1, 5)  -- столбцы в B и строки в C
    q <- chooseInt (1, 5)  -- столбцы в C
    a <- genMatrix m n
    b <- genMatrix n p
    c <- genMatrix p q
    return (a, b, c)

```

Написаны тестирующие функции для проверки свойства перемножения матриц с целыми значениями и с плавающей точкой

```
    prop_associativity_int :: Property
    prop_associativity_int =
    forAll (genAssociativeMatrices :: Gen ([[Int]], [[Int]], [[Int]])) $ \(a, b, c)
    ↪->
    matrixMultiply (matrixMultiply a b) c == matrixMultiply a (matrixMultiply b c)

    prop_associativity_double :: Property
    prop_associativity_double =
    forAll (genAssociativeMatrices :: Gen ([[Double]], [[Double]], [[Double]])) $ \(
    ↪a, b, c) ->
    let left  = matrixMultiply (matrixMultiply a b) c
    right = matrixMultiply a (matrixMultiply b c)
    in almostEqualMatrices left right 1e-9

```

2. Дистрибутивность:`A * (B + C) == A*B + A * C` для любых матриц `A`, `B` и `C`.

   Для генерации корректных матриц, подходящих под условия проверки условия написана функция `genDistributiveMatrices`

```
  genDistributiveMatrices :: (Arbitrary a) => Gen ([[a]], [[a]], [[a]])
  genDistributiveMatrices = do
  m <- chooseInt (1, 5)  -- строки в матрице A
  n <- chooseInt (1, 5)  -- столбцы в A и строки в B и C
  q <- chooseInt (1, 5)  -- столбцы в B и столбцы в C
  a <- genMatrix m n
  b <- genMatrix n q
  c <- genMatrix n q
  return (a, b, c)

```

Написаны тестирующие функции для проверки свойства перемножения матриц с целыми значениями и с плавающей точкой

```
prop_distributivity_int :: Property
prop_distributivity_int =
forAll (genDistributiveMatrices :: Gen ([[Int]], [[Int]], [[Int]])) $ \(a, b, c) ->
```

```
matrixMultiply a ( zipWith(zipWith(+)) b c) == zipWith(zipWith(+)) (matrixMultiply a
    ↪ b) (matrixMultiply a c)

prop_distributivity_double :: Property
prop_distributivity_double =
forAll (genDistributiveMatrices :: Gen ([[Double]], [[Double]], [[Double]])) $ \(a,
    ↪b, c) ->
let left  = matrixMultiply a ( zipWith(zipWith(+)) b c)
right = zipWith(zipWith(+)) (matrixMultiply a b) (matrixMultiply a c)
in almostEqualMatrices left right 1e-9


```

3. Умножение на единичную матрицу: `A * 1 == A`, где `I` — единичная матрица соответствующего размера.

   Для генерации корректных матриц, подходящих под условия проверки условия написаны функции `iMatrix` и `genMultiplicationMatrices`

```
    iMatrix :: Num a => Int -> [[a]]
    iMatrix n = [[if i == j then 1 else 0 | j <- [1..n]] | i <- [1..n]]

    genMultiplicationMatrices :: (Arbitrary a, Num a) => Gen ([[a]], [[a]])
    genMultiplicationMatrices = do
    m <- chooseInt (1, 5)  -- строки в матрице A
    n <- chooseInt (1, 5)  -- столбцы в A и строки и столбцы в I
    a <- genMatrix m n
    b <- return (iMatrix n)
    return (a, b)


```

   Написаны тестирующие функции для проверки свойства перемножения матриц с целыми значениями и с плавающей точкой

```
  prop_multiplication_int :: Property
  prop_multiplication_int =
  forAll (genMultiplicationMatrices :: Gen ([[Int]], [[Int]])) $ \(a, b) ->
  matrixMultiply a b == a

  prop_multiplication_double :: Property
  prop_multiplication_double =
  forAll (genMultiplicationMatrices :: Gen ([[Double]], [[Double]])) $ \(a, b) ->
  let left  = matrixMultiply a b
  right = a
  in almostEqualMatrices left right 1e-9

```

2.3 Функция `main`

Функция `main` является точкой входа программы, в которой выполняются тесты для проверки свойств различных функций. В частности, для каждого теста используется функция `quickCheck`, которая автоматически проверяет заданное свойство на случайных данных.

Функция представлена на листинге 3.

```
main :: IO ()
main = do
putStrLn("Тест prop_reverse:")
quickCheck prop_reverse

putStrLn("\nТест prop_change:")
quickCheck prop_change

putStrLn("\nТест prop_one:")
quickCheck prop_one

putStrLn("\n-----------------------------------")

putStrLn("\nТест prop_associativity (with Int):")
quickCheck prop_associativity_int

putStrLn("\nТест prop_associativity (with Double):")
quickCheck prop_associativity_double

putStrLn("\nТест prop_distributivity (with Int):")
quickCheck prop_distributivity_int

putStrLn("\nТест prop_distributivity (with Double):")
quickCheck prop_distributivity_double

putStrLn("\nТест prop_multiplication (with Int):")
quickCheck prop_multiplication_int

putStrLn("\nТест prop_multiplication (with Double):")
quickCheck prop_multiplication_double
```

3 Результаты работы программы

Результаты запуска программы с помощью команды `stack test` представлены ниже:

```
  Тест prop_reverse:
  +++ OK, passed 100 tests; 223 discarded.

  Тест prop_change:
  +++ OK, passed 100 tests; 423 discarded.

  Тест prop_one:
  +++ OK, passed 100 tests; 107 discarded.

  -----------------------------------

  Тест prop_associativity (with Int):
  +++ OK, passed 100 tests.

  Тест prop_associativity (with Double):
  +++ OK, passed 100 tests.

  Тест prop_distributivity (with Int):
  +++ OK, passed 100 tests.

  Тест prop_distributivity (with Double):
  +++ OK, passed 100 tests.

  Тест prop_multiplication (with Int):
  +++ OK, passed 100 tests.

  Тест prop_multiplication (with Double):
  +++ OK, passed 100 tests.
```

Для того чтобы сломать тесты необходимо изменить функцию `logBase`

```
  logBase :: Double -> Double -> Double
  logBase a b = b / log a
```

Ниже представлены ошибки при выполнении тестов

```
Тест prop_reverse:
*** Failed! Falsified (after 1 test and 6 shrinks):
0.1
0.1

Тест prop_change:
*** Failed! Falsified (after 1 test and 9 shrinks):
1.0e-2
0.1
0.1
```

```
11
12 Тест prop_one:
13 *** Failed! Falsified (after 1 test and 4 shrinks):
14 1.0e-2
```

При тестировании функции по умолчанию используется 100 тестов, чтобы увеличить количестов до 1000, необходимо использовать `quickCheckWith stdArgs {maxSuccess = 1000}`

Для проверки свойств функции `logBase` используем 1000 тестов:

```
1  putStrLn("Тест prop_reverse:")
2  quickCheckWith stdArgs {maxSuccess = 1000}  prop_reverse
3
4  putStrLn("\nТест prop_change:")
5  quickCheckWith stdArgs {maxSuccess = 1000}  prop_change
6
7  putStrLn("\nТест prop_one:")
8  quickCheckWith stdArgs {maxSuccess = 1000}  prop_one
9
```

Результат проведения 1000 тестов над свойствами функции `logBase`

```
1  Тест prop_reverse:
2  +++ OK, passed 1000 tests; 2167 discarded.
3
4  Тест prop_change:
5  +++ OK, passed 1000 tests; 4507 discarded.
6
7  Тест prop_one:
8  +++ OK, passed 1000 tests; 809 discarded.
```

Заключение

В ходе выполнения лабораторной работы были реализованы и протестированы на корректность свойства нескольких функций в языке программирования Haskell. У первой фунции, находящей логарифм числа по заданному основанию: обратное свойство, свойство смены основания, логарифм единицы. У второй функции, перемножающей матрицы: ассоциативность, дистрибутивность, умножение на единичную матрицу.

В отчете приведены результаты тестирования каждого заданного свойства для обоих функций с использованием библиотеки QuickCheck.

Список литературы

[1] Курт У. *Программируй на Haskell* / пер. с англ. С. Соловьева. — Москва: ДМК Пресс, 2019. — 384 с.