МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего 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»

Тестирование в Haskell

Практическое задание 4

Студент,
группы 5130201/20101 _____ Астафьев И. Е.

Преподаватель _____ Моторин Д. Е.

«_____»_____ 2024г.

Санкт-Петербург, 2024

Содержание

Введение

В данном отчете описаны результаты выполнения практического задания: тестирование свойств в Haskell.

Постановка задачи: Создать проект в stack. Функции записать в библиотеку Lib.hs и ограничить доступ к вспомогательным функциям. Тесты записать в Spec.hs.

Функция 1: Написать функцию сложения по модулю addMod :: Int -> Int -> Int -> Int, которая принимает три целых числа: два слагаемых и модуль, и возвращает сумму двух чисел по модулю. Используя QuickCheck, проверить следующие свойства:

1. Сложение по модулю: `(addMod x y m) mod m == (x + y) mod m`.
2. Нейтральный элемент: `addMod x 0 m == x mod m`.
3. Коммутативность: `addMod x y m == addMod y x m`.

Функция 2: Написать функцию swapTuple :: [(a, b)] -> [(b, a)], которая меняет местами элементы кортежа в списке кортежей. Используя QuickCheck, проверить следующие свойства:

1. Двойное применение функции возвращает оригинальный список кортежей: swapTuple (swapTuple [(x, y)]) == [*x, y)].
2. Если элементы списка кортежей равны, то результат будет одинаковым: swapTuple [(x, x)] == [(x, x)]
3. Результат должен содержать те же элементы для каждого кортежа из списка: fst (swapTuple [(x, y)]) == y и snd (swapTuple [(x, y)1) == x.

1 Теоретические сведения

1.1 Сложение чисел по модулю

Сложение чисел по модулю — это арифметическая операция, в которой сумма двух чисел вычисляется с последующим приведением результата к остатку от деления на заданное число (модуль). В математике это записывается следующим образом:

$$(x + y) \mod m$$

где:

- x, y — исходные числа,
- m — модуль (положительное целое число),
- $(x + y) \mod m$ — остаток от деления суммы $x + y$ на m.

Операция сложения по модулю широко используется в различных областях, таких как теория чисел, криптография, программирование, компьютерные науки и цифровая обработка сигналов.

Свойства сложения по модулю

Сложение чисел по модулю обладает рядом свойств, аналогичных обычному сложению:

1. **Коммутативность:**

   $$(x + y) \mod m = (y + x) \mod m$$

   Порядок чисел не влияет на результат.

2. **Ассоциативность:**

   $$((x + y) + z) \mod m = (x + (y + z)) \mod m$$

   Скобки можно расставлять любым образом.

3. **Существование нейтрального элемента:**

   $$(x + 0) \mod m = x \mod m$$

   Число 0 является нейтральным элементом для сложения по модулю.

4. **Закрытость:** Если $x \mod m$ и $y \mod m$ принадлежат множеству $\{0, 1, 2, \ldots, m-1\}$, то результат сложения $(x + y) \mod m$ также принадлежит этому множеству.

5. **Редукция:** Если $x, y \geq m$, то перед выполнением сложения по модулю можно привести числа к остаткам:

   $$((x \mod m) + (y \mod m)) \mod m = (x + y) \mod m$$

   Это позволяет работать с большими числами, предварительно уменьшая их размер.

Алгоритм вычисления

Для выполнения сложения по модулю выполняются следующие шаги:

1. Найти сумму чисел $x + y$.
2. Разделить полученную сумму на модуль m.
3. Найти остаток от деления, который и будет результатом.

Формула для вычисления остатка:

$$r = (x + y) - m \cdot \left\lfloor \frac{x + y}{m} \right\rfloor$$

где $\lfloor z \rfloor$ — это целая часть числа z.

Сложение по модулю используется, например, в хэш-функциях, генерации псевдослучайных чисел, шифровании и решении систем сравнений.

1.2 Перестановка элементов

Функция `swap`

Функция `swap` — это преобразование, которое принимает пару (кортеж) (a, b) и возвращает новую пару с элементами, поменянными местами: (b, a). Сигнатура:

$$\texttt{swap} :: (a, b) \to (b, a)$$

Пара (кортеж)

Пара (кортеж) в Haskell — это структура, состоящая из двух значений, которые могут иметь разные типы. Обозначается как (a, b), где a и b — типы элементов.

Список

Список в Haskell — это упорядоченная коллекция элементов одного типа. Списки обозначаются квадратными скобками, например, $[a, b, c]$. Пустой список обозначается как $[]$.

1.3 Тестирование свойств и библиотека QuickCheck

Тестирование свойств (Property Testing)

Тестирование свойств (property testing) — это подход к тестированию программ, при котором проверяются общие свойства функций вместо проверки отдельных конкретных случаев. Свойства определяются в виде логических утверждений (предикатов), которые должны выполняться для всех допустимых входных данных. Это позволяет тестировать функции на большом наборе случайных данных, а не только на заранее подготовленных примерах.

Основные преимущества:

- Автоматическое создание тестовых данных.
- Обнаружение краевых случаев (edge cases).
- Удобство выражения общих правил для проверки функций.

QuickCheck

QuickCheck — это библиотека для тестирования свойств в Haskell. Она автоматически генерирует случайные входные данные для функции и проверяет, выполняется ли заданное свойство. Если свойство нарушается, QuickCheck предоставляет контрпример, который демонстрирует, при каких входных данных функция работает некорректно.

Основные элементы QuickCheck

- **Свойства (properties):** Это логические выражения, которые описывают поведение функции. Например:

  ```
  prop_commutative :: Int -> Int -> Bool
  ```

  проверяет свойство коммутативности для сложения.

- **Класс `Arbitrary`:** Этот класс отвечает за автоматическую генерацию случайных данных. Для пользовательских типов можно определять свои правила генерации, реализуя экземпляр `Arbitrary`.

- **Функция `quickCheck`:** Основной инструмент, который запускает тестирование свойства на случайных данных. Например:

  ```
  quickCheck prop_commutative
  ```

2 Реализация программы

Полный исходный код представлен в Приложение А. Код программы.

2.1 Функция addMod

Функция `addMod` выполняет сложение двух чисел по модулю, возвращая остаток от деления их суммы на заданное значение модуля. Она принимает три параметра: два числа и модуль. Сначала вычисляется сумма чисел, затем определяется, сколько раз модуль полностью укладывается в эту сумму, после чего из суммы вычитается соответствующее количество модулей. Результатом является число в диапазоне от 0 до $m - 1$, где m — значение модуля.

2.2 Функция swapTuple

Функция `swapTuple` принимает список кортежей и возвращает новый список, где элементы каждого кортежа поменяны местами. Для этого она применяет функцию `swap`, которая меняет местами элементы одного кортежа, ко всем элементам списка, используя функцию `map`.

2.3 Тестирование

Для тестирования функций `addMod` и `swapTuple` используется библиотека `QuickCheck`, которая позволяет проверять свойства функций на больших наборах случайных данных. Основные этапы тестирования включают в себя следующие шаги:

1. Формулировка свойств: Определяются свойства, которые должна удовлетворять каждая функция. Для функции `addMod` тестируются:

   - Корректность вычисления остатка через сравнение с встроенной операцией `mod`.
   - Нейтральность прибавления нуля.
   - Коммутативность сложения.

   Для функции `swapTuple` тестируются:

   - Двойное применение возвращает исходный список.
   - Результат содержит те же элементы для каждого кортежа.
   - Каждое значение из кортежа корректно меняется местами.

2. Автоматическая генерация данных: `QuickCheck` автоматически генерирует тестовые данные для проверки свойств. Для предотвращения деления на ноль при работе с модулями используется модификатор `Positive`, который ограничивает генерацию только положительных чисел.

3. Проверка свойств: Для каждого свойства, описанного в виде функции, `QuickCheck` многократно вызывает её с разными случайными входными данными. Результаты сравниваются с ожидаемыми значениями, и если хотя бы одно из свойств не выполняется, тест завершается с сообщением об ошибке.

4. Результат тестирования: В `main` все тесты запускаются с помощью `quickCheck`, а их успешное прохождение завершается выводом строки `"ALL TESTS ARE PASSED"`.

Таким образом, тестирование подтверждает, что функции `addMod` и `swapTuple` работают корректно в большинстве случаев, предусмотренных их контрактом.

3 Результаты

Результаты прохождения тестов, описанных выше, представлены на Рис. 1.

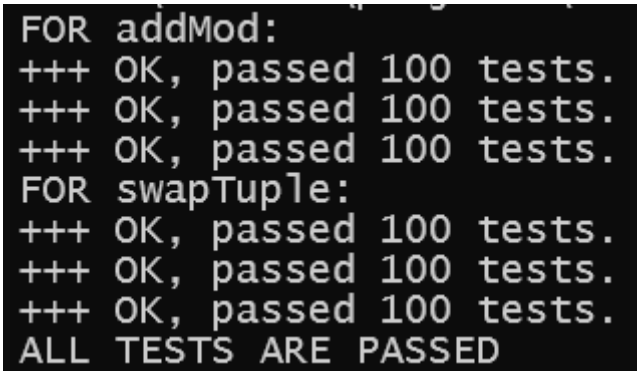

Рис. 1. Результаты прохождения тестов

Были написаны тесты и проверены с помощью QuickCheck. Было проверено, что функции удовлетворяют свойствам, описанным в постановке задачи к практической работе. По результатам работы программы, можно увидеть, что все тесты пройдены успешно.

Заключение

В ходе выполнения данного отчета были успешно реализованы две функции на языке программирования Haskell: функция сложения по модулю addMod и функция обмена элементов кортежа в списке swapTuple. Для каждой из этих функций было проведено тестирование свойств с использованием библиотеки QuickCheck. Все проверяемые свойства показали корректную работу функций при различных входных данных.

Таким образом, поставленные задачи выполнены полностью: созданы необходимые функции, ограничены права доступа к вспомогательным функциям, проведены тесты и подтверждена правильность работы функций через проверку их свойств.

Приложение А. Код программы

Lib.hs

```haskell
module Lib
    ( addMod
    , swapTuple
    ) where

addMod :: Int -> Int -> Int -> Int
addMod firstSummand secondSummand modulo = sum - modulo * floor(fromIntegral sum /
↪  fromIntegral modulo)
    where sum = firstSummand + secondSummand

swap :: (a, b) -> (b, a)
swap (a, b) = (b, a)

swapTuple :: [(a, b)] -> [(b, a)]
swapTuple tupleList = map swap tupleList
```

Spec.hs

```haskell
import Lib
import Test.QuickCheck

-- без Positive - исключение деление на 0 (даже на `mod`)
prop_withBuiltinAddMod :: Int -> Int -> Positive Int -> Bool
prop_withBuiltinAddMod x y (Positive m) = (addMod x y m) `mod` m == (x + y) `mod` m

prop_neutralElement :: Int -> Positive Int -> Bool
prop_neutralElement x (Positive m) = addMod x 0 m == x `mod` m

prop_commutativity :: Int -> Int -> Positive Int -> Bool
prop_commutativity x y (Positive m) = addMod x y m == addMod y x m

prop_doubleApply :: (Eq a, Eq b, Arbitrary a, Arbitrary b) => [(a, b)] -> Bool
prop_doubleApply xs = swapTuple (swapTuple xs) == xs

prop_sameElements :: (Eq a, Arbitrary a) => [a] -> Bool
prop_sameElements xs = swapTuple (zip xs xs) == zip xs xs

prop_swapEqualityElements :: (Eq a, Eq b, Arbitrary a, Arbitrary b) => [(a, b)] -> Bool
prop_swapEqualityElements xs = all checkSwap xs
  where
    checkSwap (x, y) =
      let swapped = swapTuple [(x, y)]
      in fst (head swapped) == y && snd (head swapped) == x

main :: IO ()
main = do
```

```
29      putStrLn "FOR addMod:"
30      quickCheck prop_withBuiltinAddMod
31
32      quickCheck prop_neutralElement
33
34      quickCheck prop_commutativity
35
36      putStrLn "FOR swapTuple:"
37
38      quickCheck (prop_doubleApply :: [(Int, Char)] -> Bool)
39
40      quickCheck (prop_sameElements :: [Char] -> Bool)
41
42      quickCheck (prop_swapEqualityElements :: [(Int, Char)] -> Bool)
43
44      putStrLn "ALL TESTS ARE PASSED"
```