CS314. Функциональное программирование

Лекции 1–2. Введение в функциональное программирование на языке Haskell

В. Н. Брагилевский

Направление «Фундаментальная информатика и информационные технологии»
Институт математики, механики и компьютерных наук имени И. И. Воровича
Южный федеральный университет

3 сентября 2016 г.

Функциональные языки программирования

- Lisp и его наследники (Common Lisp, Scheme, Clojure)
- Haskell
- ML и его диалекты (Standard ML, OCaml)
- F#
- Scala
- Erlang
- ...

Важнейшие черты функционального стиля

- Всякое вычисление трактуется как вычисление значения математической функции.
- Отсутствует изменяемое состояние (нет оператора присваивания, переменных, циклов).
- Имеется богатый инструментарий для работы с функциями (функции высших порядков, различные способы определения функций).

Введение в функциональное программирование

1. Язык Haskell и среда программирования
2. Пример 1: сумма чисел от 1 до n
3. Коротко о типах
4. Пример 2: вычисление квадратного корня
5. Пример 3: вычисление кубического корня
6. Функции высшего порядка
7. Пример 4: часто встречающиеся слова

Содержание

Язык программирования Haskell

Определение из википедии

Haskell — это чисто функциональный язык общего назначения с нестрогой семантикой, сильной статической типизацией и выводом типов.

Haskell Curry (1900–1982)

Создатели языка (1990)

Simon Peyton Jones, Lennart Augustsson, Dave Barton, Brian Boutel, Warren Burton, Joseph Fasel, Kevin Hammond, Ralf Hinze, Paul Hudak, John Hughes, Thomas Johnsson, Mark Jones, John Launchbury, Erik Meijer, John Peterson, Alastair Reid, Colin Runciman, Philip Wadler.

Стандарты и реализации

Версии и стандарты

- Haskell 1.0, 1.1, 1.2, 1.3, 1.4 (1990 — 1997)
- Haskell 98
- Haskell 2010
- Haskell 2020

Реализации

- **GHC (Glasgow Haskell Compiler)**
- UHC (Utrecht Haskell Compiler)
- YHC (York Haskell Compiler)
- LHC (LLVM Haskell Compiler)
- Hugs (только интерпретатор)
- Расширения — extensions

Haskell Platform: http://www.haskell.org/platform/

- Компилятор GHC и интерпретатор GHCi (8.0.1).
- Система сборки Cabal.
- Система управления проектами Stack.
- Средства профилирования кода.
- Библиотеки (в полной версии).

Содержание

Сумма чисел

Постановка задачи

Дано число n. Вычислить сумму:

$$1 + 2 + \cdots + n.$$

Идеи решения

- Рекурсивное решение: сумма(n) = n + сумма($n-1$).
- Итеративное решение: начиная с нуля, последовательно прибавляем числа от 1 до n.

Рекурсивное решение

Код

```
sumN 0 = 0
sumN n = n + sumN (n-1)
```

Вычисление

```
sumN 5 = 5 + sumN 4
       = 5 + (4 + sumN 3)
       = 5 + (4 + (3 + sumN 2))
       = 5 + (4 + (3 + (2 + sumN 1)))
       = 5 + (4 + (3 + (2 + (1 + sumN 0))))
       = 5 + (4 + (3 + (2 + (1 + 0))))
       = 5 + (4 + (3 + (2 + 1)))
       = 5 + (4 + (3 + 3))
       = 5 + (4 + 6)
       = 5 + 10
       = 15
```

Итеративное решение

Код

```
sumN' 0 = 0
sumN' n = iter n 0
  where
    iter 0 s = s
    iter i s = iter (i-1) (s+i)
```

Вычисление

```
sumN' 5 = iter 5 0
        = iter 4 5
        = iter 3 9
        = iter 2 12
        = iter 1 14
        = iter 0 15
        = 15
```

NB!

Рекурсия на уровне кода превратилась в итерацию на этапе вычислений.

Что там с типами?

```
ghci> :t sumN
sumN :: (Eq a, Num a) => a -> a
ghci> :t sumN'
sumN' :: (Eq a, Num a) => a -> a
```

Содержание

Типы: основные понятия

- Любое выражение имеет тип (Int, Integer, Double, Bool, Char).
- В языке Haskell используется сильная статическая типизация (strong static typing) с выводом типов (type inference).
- Для справки:
  - Паскаль, Java — сильная статическая;
  - C — слабая статическая;
  - PHP, Javascript — слабая динамическая;
  - Python, Ruby — сильная динамическая.
- Замечание: любая простая классификация условна.

Определение типа в GHCi

```
ghci> :t 'a'
'a' :: Char
ghci> :t True
True :: Bool
ghci> :t 42
42 :: Num a => a
ghci> :t 42.234
42.234 :: Fractional a => a
ghci> :t max
max :: Ord a => a -> a -> a
ghci> :t succ
succ :: Enum a => a -> a
```

Типы и классы типов

- Значения имеют тип.
- Тип определяет множество значений.
- Класс типов определяет набор функций, которые можно вызывать для значений.
- Типы могут принадлежать классам типов (иметь экземпляр класса).

Некоторые классы типов

- `Eq` — проверка на равенство и неравенство.
- `Ord` — упорядочение (меньше, больше и т.д.).
- `Enum` — последовательность (следующий, предыдущий).
- `Num` — числовые операции (сложение, вычитание и др.).
- `Integral` — операции и функции для целых чисел (div, mod).
- `Fractional` — операции и функции для дробных чисел (деление).
- `Floating` — операции и функции для вещественных чисел (sin).
- `Show` — преобразование значения типа в строку (show).

Пример: классы, которым принадлежит тип Integer

`Eq`, `Ord`, `Num`, `Enum`, `Integral`, `Show`

Примеры: классы типов Eq и Show

Класс Eq

```
ghci> :t (==)
(==) :: Eq a => a -> a -> Bool
ghci> :t (/=)
(/=) :: Eq a => a -> a -> Bool
ghci> 5 == 5
True
ghci> False == True
False
ghci> 'x' /= 'y'
True
```

Класс Show

```
ghci> :t show
show :: Show a => a -> String
ghci> show 42
"42"
ghci> show pi
"3.14159265358979 3"
ghci> show True
"True"
```

- Классы типов реализуют ограниченный полиморфизм.

Простейшие структуры данных: кортежи и списки

```
type SomeData = (Double, Integer, Bool) -- кортеж
type Ages = [Integer] -- список
```

- Гомогенные (список) и гетерогенные (кортеж) структуры данных.
- Рекурсивная природа списка: голова и хвост (тоже список).

Пример 1: решение на списках

```
sum' :: Num a => [a] -> a
sum' [] = 0
sum' (x:xs) = x + sum' xs

sumN'' n = sum' [1..n]
```

NB!

- Образцы для списка.
- Рекурсия по структуре списка.
- Генератор списка.

Содержание

Вычисление квадратного корня

Постановка задачи

Вычислить квадратный корень из заданного положительного вещественного числа.

Дано: $y > 0$. Найти $x(> 0) : \ x^2 = y$.

Идея решения (метод Ньютона)

Если x — некоторое приближение к $\sqrt{y}$, то более точное приближение можно вычислить по формуле:

$$x' = \frac{x + y/x}{2}$$

Можно выбрать произвольное начальное приближение, а затем улучшать его, пока оно не окажется достаточно хорошим.

Решение

```
abs' a = if a > 0 then a else -a
sqr' a = a * a
average a b = (a + b) / 2
eps = 0.00000000001

sqrt' y = sqrtIter 1
  where
    goodEnough x = abs' (sqr' x - y) < eps
    improve x = average x (y/x)
    sqrtIter guess
        | goodEnough guess = guess
        | otherwise = sqrtIter (improve guess)
```

Решение на Common Lisp

```
(defun abs1 (a) (if (< a 0) (- a) a))
(defun sqr (a) (* a a))
(defun average (a b) (/ (+ a b) 2))
(defconstant eps 0.00000000001)

(defun sqrt1 (y)
  (labels (
    (goodEnough (x) (< (abs1 (- (sqr x) y) ) eps))
    (improve (x) (average x (/ y x)))
    (sqrtIter (guess) (
      if (goodEnough guess)
         guess
         (sqrtIter (improve guess)))))
    (sqrtIter 1)
    )
  )
```

Решение на Scala (1)

```
def abs(a: Double): Double = {
  if (a > 0) a else -a
}

def sqr(a: Double): Double = {
  a * a
}

def average(a: Double, b: Double): Double = {
  (a + b) / 2
}

def eps = 0.00000000001
```

Решение на Scala (2)

```
def sqrt1(y: Double): Double = {
  def goodEnough(x: Double): Boolean = {
    abs(sqr(x) - y) < eps
  }
  def improve(x: Double): Double = {
    average(x, y/x)
  }
  def sqrtIter(guess: Double): Double = {
    if (goodEnough(guess))
      guess
    else
      sqrtIter(improve(guess))
  }
  sqrtIter(1)
}
```

Содержание

Вычисление кубического корня

Постановка задачи

Вычислить кубический корень из заданного положительного вещественного числа.
Дано: $y > 0$. Найти $x(>0): \ x^3 = y$.

Идея решения (метод Ньютона)

Если x — некоторое приближение к $\sqrt[3]{y}$, то более точное приближение можно вычислить по формуле:

$$x' = \frac{2x + y/x^2}{3}$$

Можно выбрать произвольное начальное приближение, а затем улучшать его, пока оно не окажется достаточно хорошим.

Идея

Вычисление квадратного корня

```
abs' a = if a > 0 then a else -a
sqr' a = a * a
average a b = (a + b) / 2
eps = 0.00000000001

sqrt' y = sqrtIter 1
  where
    goodEnough x = abs' (sqr' x - y) < eps
    improve x = average x (y/x)
    sqrtIter guess
        | goodEnough guess = guess
        | otherwise = sqrtIter (improve guess)
```

NB!

Функцию improve и возведение в квадрат нужно параметризовать!

Содержание

Функции высшего порядка — кто они?

Определение (Википедия)

Функция высшего порядка (higher-order function) — функция, принимающая функции в качестве аргументов или возвращающая функцию в качестве результата.

Пример

```
twice :: (a -> a) -> a -> a
twice f x = f (f x)
```

```
ghci> twice sqrt 16
2.0
```

Функция max и частичное применение

```
ghci> :t max
max :: Ord a => a -> a -> a
ghci> max 4 5
5
ghci> (max 4) 5
5
```

- Вопрос: каков тип выражения max 4?

```
ghci> :t max 4
max 4 :: (Ord a, Num a) => a -> a
```

Каррированные функции и частичное применение

- Все функции в языке Haskell каррированы, т.е. они могут принимать только один параметр.
- Функции с несколькими параметрами при передаче им одного параметра возвращают новую функцию от меньшего числа параметров.
- Вызов функции с недостаточным числом параметров называется частичным применением (partial application).

Пример

```
multThree :: Int -> Int -> Int -> Int
multThree x y z = x * y * z
```

```
multThree :: Int -> (Int -> (Int -> Int))
```

```
ghci> let multTwoWithNine = multThree 9
ghci> multTwoWithNine 2 3
54
```

Композиция функций

Композиция функций в математике

$$(f \circ g)(x) = f(g(x))$$

Композиция функций в Haskell

```
ghci> (negate . abs) (-5)
-5
ghci> (negate . abs) 5
-5
ghci> negate (abs 5)
-5
ghci> :t (negate . abs)
(negate . abs) :: Num c => c -> c
```

Сечения арифметических операций

```
ghci> 3 + 5
8
ghci> (3+) 5
8
ghci> (+5) 3
8
ghci> :t (3+)
(3+) :: Num a => a -> a
ghci> :t (+5)
(+5) :: Num a => a -> a
```

```
ghci> twice (+3) 1
7
```

Сечения арифметических операций: вычитание

```
ghci> :t (5-)
(5-) :: Num a => a -> a
ghci> :t (-5)
(-5) :: Num a => a
ghci> :t (subtract 5)
(subtract 5) :: Num a => a -> a
```

```
ghci> (subtract 5) 47
42
ghci> (47-) 5
42
```

Сечения функций

```
ghci> :t elem
elem :: Eq a => a -> [a] -> Bool
```

```
isUpperAlpha :: Char -> Bool
isUpperAlpha = (`elem` ['A'..'Z'])
```

```
ghci> isUpperAlpha 'X'
True
ghci> isUpperAlpha 'a'
False
```

Пример

Задача

Комиссия за выполнение платежа в банке «Край-Финанс» составляет 1% от суммы, причём минимальный размер комиссии — 30 рублей. Вычислить размер комиссии по заданной сумме платежа.

```
comission :: Double -> Double
comission = max 30 . (* 0.01)
```

```
ghci> comission 1970
30.0
ghci> comission 4570
45.7
```

Способы создания функций «на лету»

- Частичное применение
- Композиция
- Сечения
- ???

Анонимные функции

```
ghci> (\x -> x * x + x) 5
30
ghci> (\xs -> 2 * length xs) [1,3,5]
6
ghci> (\(x:xs) -> (x, length xs + 1)) [1,1,1,1]
(1,4)
ghci> (\x y -> x^2 + y^2) 2 3
13
ghci> (\x -> if odd x then 1 else 0) 5
1
```

```
ghci> :t (\x y -> x^2 + y^2)
(\x y -> x^2 + y^2) :: Num a => a -> a -> a
```

Происхождение идеи: λ-функции (Алонсо Чёрч)

- $\lambda x.x+1$ — это функция, прибавляющая единицу к своему аргументу ($f(x) = x+1$, $x \mapsto x+1$);
- $\lambda x.\lambda y.x+y$ — это функция, складывающая два своих аргумента ($g(x,y) = x+y$, $(x,y) \mapsto x+y$).

Композиция как ФВП

```
(.) :: (b -> c) -> (a -> b) -> a -> c
f . g = \x -> f (g x)
```

Функция ($)

```
($) :: (a -> b) -> a -> b
f $ x = f x
```

- Всё дело в ~~волшебных~~ приоритете!

```
ghci> sin 4 + 5
4.243197504692072
ghci> sin (4 + 5)
0.4121184852417566
ghci> sin $ 4 + 5
0.4121184852417566
ghci> not $ odd $ 3 + 7
True
```

Композиция vs ($)

```
f = ceiling . negate . tan . cos . max 50
```

```
f x = ceiling (negate (tan (cos (max 50 x))))
```

```
f x = ceiling $ negate $ tan $ cos $ max 50 x
```

```
ghci> f 42
-1
```

Возвращение к примеру 3

```
newton :: (Double -> Double -> Double) -- improve
          -> (Double -> Double) -- проверка
          -> Double -> Double -- аргумент и результат
```

```
newton improve check y = iter 1
  where
    goodEnough x = abs' (check x - y) < eps
    iter guess
        | goodEnough guess = guess
        | otherwise = iter (improve guess y)

sqrt' = newton (\x y -> average x (y/x)) (^2)
cbrt = newton (\x y -> (2*x+y/(x^2))/3) (^3)
```

Ещё одно решение

Общие идеи метода Ньютона

- Начальное приближение.
- Имеется способ улучшать приближение (improve) $\Longrightarrow$ список приближений, вообще говоря бесконечный.
- Имеется способ находить подходящее приближение (goodEnough).

```
newton improve check y = findFirst goodEnough (guesses 1)
 where
  findFirst pred xs = undefined -- найти первый элемент,
                                -- удовлетворяющий предикату
  goodEnough x = undefined -- подходит ли приближение?
  guesses x = undefined -- список приближений
```

Решение: метод Ньютона на списках

```
newton' improve check y = findFirst goodEnough (guesses 1)
 where
  goodEnough x = abs (check x - y) < eps
  guesses x = x : guesses (improve x y)

findFirst pred [] = error "no such element"
findFirst pred (x:xs)
   | pred x = x
   | otherwise = findFirst pred xs
```

- `newton'`, `findFirst` — функции высшего порядка.
- Решение можно значительно улучшить, воспользовавшись стандартными функциями обработки списков (`filter`, `iterate`).

Содержание

Часто встречающиеся слова

Постановка задачи

Дан текст. Найти 50 наиболее часто повторяющихся в этом тексте слов.

Компоненты решения

- Текст — это список символов (строка, String).
- Разбиение на слова $\Longrightarrow$ список слов.
- Подсчёт числа повторений $\Longrightarrow$ список пар (слово, количество).
- Выборка и печать наиболее часто встречающихся слов.

Цель

frequentWords :: **String** −> **String**

Решение

```
frequentWords :: String -> String
frequentWords = showWords
                . takeFrequent
                . countRepetitions
                . extractWords

extractWords :: String -> [String]
countRepetitions :: [String] -> [(String, Integer)]
takeFrequent :: [(String, Integer)] -> [String]
showWords :: [String] -> String
```

$$(f \circ g)(x) = f(g(x))$$

Итоги

- Язык Haskell.
- Функциональное программирование — без присваиваний и циклов.
- Рекурсивные и итеративные вычисления.
- Типы и классы типов.
- Простейшие структуры данных: списки и кортежи.
- Функции высших порядков и определение функций «на лету».