Функциональное программирование с зависимыми типами на языке Idris

Лекция 1. Введение в Idris, элементы функционального программирования

В. Н. Брагилевский

21 ноября 2017 г.

Факультет компьютерных наук, НИУ «Высшая школа экономики»

Институт математики, механики и компьютерных наук имени И. И. Воровича, Южный федеральный университет (Ростов-на-Дону)

Контакты

Виталий Брагилевский bravit111@gmail.com

- _bravit (ru)
- 💟 VBragilevsky (en)
- √ bravit111
- bravit
- _ bravit



Функциональные языки программирования

- Lisp и его наследники (Common Lisp, Scheme, Clojure)
- ML и его диалекты (Standard ML, OCaml), F#
- · Haskell, Agda, Idris
- 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).

ldris: общая характеристика

Язык программирования Idris

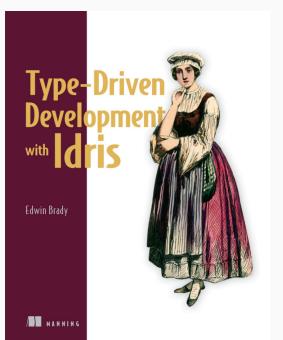
Idris — это компилируемый, чисто функциональный язык общего назначения с зависимыми типами и строгим вычислением.



Edwin Brady, University of St Andrews

Инструментарий

- Сайт: https://www.idris-lang.org/
- Текущая версия: 1.1.1 (5 августа 2017 года), версия 1.0 вышла 1 апреля 2017 года
- Компилятор
- Интерпретатор в стиле REPL: Read/Eval/Print/Loop
- Поддержка в редакторах: Atom, vi, Emacs



Edwin Brady,
Type-Driven
Development with
Idris, Manning
Publications, 2017

Примеры программирования в функциональном стиле

Сумма чисел

Постановка задачи

Дано число *п*. Вычислить сумму:

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

Идеи решения

- Решение 1 (рекурсивное): сумма(n) = n +сумма(n-1).
- Решение 2 (итерационное): заводим аккумулятор с нулевым значением и последовательно прибавляем к нему числа от 1 до *n*.

Решение 1

Код

```
sumN : Integer -> Integer
sumN 0 = 0
sumN n = n + sumN (n - 1)
```

Вычисление

NB!

Рекурсивный sumN 5 = 5 + sumN 4 вычислительный процесс.

$$= 5 + (4 + sumN 3)$$

= $5 + (4 + (3 + sumN 2))$

$$= 5 + (4 + (3 + (2 + sumN 1)))$$

$$= 5 + (4 + (3 + (2 + (1 + sumN 0))))$$

$$= 5 + (4 + (3 + (2 + 1)))$$

$$= 5 + (4 + (3 + 3))$$

$$= 5 + (4 + 6) = 5 + 10 = 15$$

Решение 2

Код

```
sumN' : Integer -> Integer
sumN' 0 = 0
sumN' n = go 0 n
    where
        go s 0 = s
        go s i = go (s + i) (i - 1)
```

Вычисление

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

NB!

Рекурсия в коде превратилась в итерационный вычислительный процесс.

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' : (Ord a, Neq a) \Rightarrow a \rightarrow a
abs' a = if a > 0 then a else -a
sgr' : Num a => a -> a
sar' a = a * a
average : Double -> Double -> Double
average ab = (a + b) / 2
eps : Double
eps = 0.00000000001
```

```
sqrt' : Double -> Double
sqrt' y = qo 1
 where
    goodEnough : Double -> Bool
    goodEnough x = abs' (sqr' x - y) < eps
    improve : Double -> Double
    improve x = average x (y/x)
    go : Double -> Double
    go guess = if goodEnough guess then guess
               else qo (improve quess)
```

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)))
    (go (quess) (
      if (goodEnough guess)
         quess
         (qo (improve quess)))))
    (qo 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
```

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

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.

Вычисление квадратного корня

```
sqrt' : Double -> Double
sqrt' y = go 1
```

NB!

Функцию improve и возведение в квадрат нужно параметризовать!

<u>where</u>

Метод Ньютона: общее решение

```
newton : (Double -> Double -> Double) -- improve
          -> (Double -> Double) -- check
          -> Double -> Double
newton improve check y = qo 1
  where
    goodEnough : Double -> Bool
    goodEnough x = abs' (check x - y) < eps
    go : Double -> Double
    go guess = if goodEnough guess then guess
               else qo (improve quess y)
```

- Анонимные функции
- Частичное применение

Пример «настоящей» функциональной программы

```
average : (str : String) -> Double
average str = let num words = word count str
                 total length = sum (word lengths (words str))
              in cast total length / cast num words
 where
   word count : String -> Nat
   word count str = length (words str)
   word lengths: List String -> List Nat
   word lengths strs = map length strs
showAverage : String -> String
showAverage str = "Средняя длина: " ++ show (average str)
                  ++ "\n"
main : IO ()
main = repl "Введите строку: " showAverage
```

module Main

Важнейшие команды интерпретатора

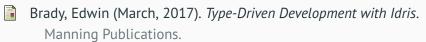
- :type ???
- :doc ???
- :exec ???
- :load ???
- · :reload
- :search ???
- :quit

```
Idris> :search Double -> Double
```

- = Prelude.Doubles.acos : Double -> Double
- = Prelude.Doubles.asin : Double -> Double

. . .

Список литературы



Idris: A Language with Dependent Types. URL: http://www.idris-lang.org/.

Харольд Абельсон, Джеральд Сассман (1985). *Структура и интерпретация компьютерных программ*.