ФУНКЦИОНАЛЬНОЕ ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Санкт-петербургский государственный политехнический университет
Институт Компьютерных Наук и Технологий
Кафедра: Информационные и Управляющий Системы
Автор: Лукашин Антон Андреевич

2016-2017

Содержание

- Lambda-операции
  - *sort*
  - *reduce, reduceLeft, reduceRight*
  - *foldLeft, foldRight*
- Циклы
  - *while*
  - *for*
  - *foreach*
- Unit
- Монады

Sort

- Sorted
  - *метод, сортирующий коллекцию*
  - *Сортировка определяется для типа элементов коллекции*
  - *Типа элементов класса должен быть расширен trait Ordered*
- sortWith(t1:T, t2:T -> boolean)
  - *Принимает функцию, превращающую два параметра в булевое значение*
  - *Возвращает сортированную коллекцию (новую!)*

reduceLeft, reduceRight, reduce

- **reduceLeft**[B >: A](f: (B, A) ⇒ B): B
  - *op(op(… op(x_1, x_2) …, x_{n-1}), x_n)*
  - *Применяет бинарный оператор ко всем элементам слева направо*
- **reduceRight**[B >: A](op: (A, B) ⇒ B): B
  - *op(x_1, op(x_2, …, op(x_{n-1}, x_n)…))*
  - *Применяет бинарный оператор ко всем элементам справа налево*
- **reduce**[A1 >: A](op: (A1, A1) ⇒ A1): A1
  - *Применяет бинарный оператор ко всем элементам*
  - *Порядок не определен*
  - *Оператор должен обладать свойством ассоциативности*

foldLeft, foldRight, fold

- **foldLeft**[B](z: B)(op: (B, A) ⇒ B): B
  - *op(...op(z, x_1), x_2, ..., x_n)*
  - *Применяет бинарный оператор ко всем элементам слева направо, начиная с изначального значения*
- **foldRight**[B](z: B)(op: (A, B) ⇒ B): B
  - *op(x_1, op(x_2, ... op(x_n, z)...))*
  - *Применяет бинарный оператор ко всем элементам справа налево, начинаяс изначального значения*
- **fold**[A1 >: A](z: A1)(op: (A1, A1) ⇒ A1): A1
  - *Применяет бинарный оператор ко всем элементам, начиная с изначального значения*
  - *Порядок не определен*
  - *Оператор должен обладать свойством ассоциативности*

Имплементация «левых» функций

- def reduceLeft(op: (T,T) => T) : T = this match {

  case Nil => throw new Error(“Nil.reduceLeft is not defined”)

  case firstElement :: tail => (tail foldLeft firstElement)(op)

}

- def foldLeft(z:U)(op: (U, T) => U):U = this match {

  case Nil => z

  case firstElement :: tail => (tail foldLeft op(z, x))(op)

}

Имплементация «правых» функций

- def reduceRight(op: (T,T) => T): T = this match {

```
case Nil => throw new Error(“Nil.reduceLeft is not defined”)
case firstElement :: Nil => firstElement
case firstElement :: tail => op(firestElement, tail.reduceRight(op))
```

}

- def foldRight(z:U)(op: (T,U)=>U):U this match {

```
case Nil => z
case firstElement :: tail => op(firstElement, (tail foldRight z)(op))
```

}

Имплементация Reverse с помощью foldLeft

- def reverse[T] (xs: List[T]):List[T] = ???
- Первоначальная имплементация (Lection-3.1) имела линейную сложность
- Предлагается реализовать данную функцию с применением foldLeft
- def reverse[T] (xs: List[T]):List[T] = (xs foldLeft z)(op)
- z и op необходимо определить
- Для получения ответа рассмотрим 3 шага
  - *Nil*
  - *List(x)*
  - *Произвольный список*

Имплементация Reverse с помощью foldLeft

- reverse(Nil) == Nil
- Nil

Имплементация Reverse с помощью foldLeft

- reverse(Nil) == Nil
- Nil
- reverse(Nil)

Имплементация Reverse с помощью foldLeft

- reverse(Nil) == Nil
- Nil
- reverse(Nil)
- (Nil foldLeft z?)(op?)

Имплементация Reverse с помощью foldLeft

- reverse(Nil) == Nil
- Nil
- =reverse(Nil)
- =(Nil foldLeft z?)(op?)
- =z

Имплементация Reverse с помощью foldLeft

- reverse(Nil) == Nil
- Nil
- =reverse(Nil)
- =(Nil foldLeft z?)(op?)
- =z?
- Z?=List()

Имплементация Reverse с помощью foldLeft

- Для вычисления op? рассмотрим List(x)
- List(x)

Имплементация Reverse с помощью foldLeft

- Для вычисления op? рассмотрим List(x)
- List(x)
- =reverse(Lit(x))

Имплементация Reverse с помощью foldLeft

- Для вычисления op? рассмотрим List(x)
- List(x)
- =reverse(Lit(x))
- =(List(x) foldLeft Nil)(op?)

Имплементация Reverse с помощью foldLeft

- Для вычисления op? рассмотрим List(x)
- List(x)
- =reverse(Lit(x))
- =(List(x) foldLeft Nil)(op?)
- =op?(Nil, x)

Имплементация Reverse с помощью foldLeft

- Для вычисления op? рассмотрим List(x)
- List(x)
- =reverse(Lit(x))
- =(List(x) foldLeft Nil)(op?)
- =op?(Nil, x)
- =List(x) = x :: List()

Имплементация Reverse с помощью foldLeft

- def reverse[T] (xs: List[T]):List[T] =

  (xs foldLeft List[T]()) ((xs,x)=>x :: xs)

- Какова сложность данной реализации?

while, forEach

- Базовый цикл с предусловием, возвращающий Unit

```
while(condition) {
        expressions
}
```

- Базовый цикл с постусловием

```
do {
        expressions
} while(condition)
```

- **def foreach(f: (A) ⇒ Unit): Unit**
  - Применяет функцию f ко всем элементам коллекции

for

- for loops with ranges
  - *for(I <- 1 to 10) {println i}*
  - *for(I <- 1 until 10) {println i]*
  - *for(I <- 1 to 5; y <- 1 to 3) {println("%s - %s").format(I,y)}*
  - *for(I <- 1 to 10; if I !=4; if I % 2 == 0) {println i}*
  - *for(I <- 1 to 5) yield i*2*
- for loops with collections
  - *for(x <- List(1,2,3))*
- for with futures

```
val result1 = future(...); val result2 = future(...)
val res = for {
  r1 <- result1
  r2 <- result2
} yield (r1+r2)
```

Монады

- Монада - это

Спасибо за внимание!

- Планы на следующую лекцию
  - *Монады*