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

- Подстановка типов (Substitution Principle)
  - *Типы данных*
  - *Функция как объект*
  - *Принцип подстановки Барбары Лисков (*Liskov Substitution Principle*)*
  - *Соответствие типов*
- Некоторые свойства списков
  - ++
  - *reverse*
- Теория категорий и функциональное программирование

А где же монады?

Categories

Constructions

Functors

Natural Transformations

Algebras and Initial Algebras

Monads and Kleish Triples

Функция как объект

- A => B эквивалентно Function1[A, B]

```
trait Function1[A, B] {
   def apply(x: A): B
}
```

- Также описаны Function2, Fucntion3… Function22
- Анонимные функции (x: Int) => x*x эквивалетны

```
{ class AnonFun extends Function1[Int, Int] {
      def apply(x: Int) = x * x
   }
   new AnonFun
}
```

Функция и метод

- def f(x: Int): Boolean = … - является методом и не является функцией
- Автоматическая конвертация при использовании

```
val list:List[Int] = List(1,2,3)

list.filter(f)


new Function1[Int, Boolean] {

   def apply(x: Int) = f(x)

}
```

Наследование и полиморфизм

- Ограничения типов (bounds)
- Вариативность типов (variance)

Ограничение типов

- Определим метод assertAllElementsPositive(IntSet)
  - *Возвращает само множество, если условие выполняется*
  - *В противном случае выбрасывает исключение*
- Каким будет лучший тип, описывающий результат выполнения этого метода?
  - *IntSet*
  - *Или что-то более точное? Например, NatSet(натуральные числа)*
- Ограничения типов:
  - S <: T, что означает - S *подтип* T
  - S >: T, что означает S *супертип* T, или T *подтип* S.
  - Можно ограничивать тип сверху и снизу

Ковариантность типов

- NonEmpty <: IntSet (непустые множества являются подножеством произвольного множества целых чисел)
- List[NonEmpty] <: List[IntSet] ???

Ковариантность типов

- NonEmpty <: IntSet (непустые множества целых чисел являются подмножеством произвольного множества целых чисел)
- List[NonEmpty] <: List[IntSet]
- Интуитивно – да (список непустых множеств целых чисел является более специфическим случаем списка произвольных множеств целых чисел)
- Типы, для которых это выполняется называются ковариантными, так как их отношение сопоставляется по типовому параметру
- Обладают ли все типы свойство ковариантности?
  - *Set ?*
  - *Array ?*

Проблемы типизирования массива

- Рассмотрим следующий Java код

N o n E m p t y [] a = new N o n E m p t y []{ new N o n E m p t y (1 , Empty , E m p t y)}

I n t S e t [] b = a

b [0] = E m p t y

N o n E m p t y s = a [0]

- На последней строке мы присваиваем Empty переменной типа NonEmpty!
- Рассмотрим Scala код

val a: Array[NonEmpty] = Array(new NonEmpty(1, Empty, Empty))

val b: Array[IntSet] = a

b(0) = Empty

val s: NonEmpty = a(0)

- Код аналогичный, однако какой будет результат?

Проблемы типизирования массива

- Рассмотрим следующий Java код

  N o n E m p t y [] a = new N o n E m p t y []{ new N o n E m p t y (1 , Empty , E m p t y)}

  I n t S e t [] b = a

  b [0] = E m p t y - *ArrayStateException*

  N o n E m p t y s = a [0]

- Рассмотрим Scala код

  val a: Array[NonEmpty] = Array(new NonEmpty(1, Empty, Empty))

  val b: Array[IntSet] = a *- Type error*

  b(0) = Empty

  val s: NonEmpty = a(0)

- Таким образом мы видим, что некоторые типы должны быть ковариантны, тогда как другие - нет

Вариативность(variance)

- C[T] – параметризованный тип
- A, B – типы, для которых выполянется A <: B
- Тогда для C[T] возможны следующие варианты:
  - *C[A] <: C[B]* - *ковариантность*
  - *C[A] >: C[B]* - *контрвариантность*
  - *C[A] и C[B] не являются подтипами друг-друга*
- Scala позволяет определять вариантность типов
  - *class C[+A]* - *ковариатность*
  - *class C[-A]* - *контрвариантность*
  - *class C[A]* - *невариантность*

Принцип подстановки Лисков

- Барбара Лисков (07.11.1939) – Лос-Анджелес
- Дедушка и бабушка (Лев Губерман и Роза Марголис) – эмигранты из Российской Империи
- Руководила разработкой языков Клу(CLU, первые абстрактные типы) и Argus
- Объектно-ориентированная СУБД Thor
- Принцип подстановки
- С 1972 года работает и преподает в MIT
- Награды
  - *Медаль фон Неймана*
  - Почётный докторский титул от Швейцарской высшей технической школы Цюриха
  - *Премия Тьюринга*
  - *Премия Гарольда Пендера*

Принцип подстановки Лисков

- *«Пусть q(x) является свойством верным относительно объектов x некоторого типа T. Тогда q(y) также должно быть верным для объектов y типа S, где S является подтипом типа T.»*
- *Если* A <: B*, тогда все что можно выполнять над значением типа* B *должно также выполняться над значением типа* A
- *Или иначе: наследующий класс должен дополнять, а не замещать поведение базового класса*
- *Или иначе:* Если у нас есть класс A и расширяющий его класс B, то возможно замещение использования класса A на B, при этом функциональность программы должна сохраниться
- Задача
  - *type A = IntSet => NonEmpty*
  - *Type B = NonEmpty => IntSet*
  - *Руководствуясь правилом Лисков определить отношения A к B*

Принцип подстановки Лисков

- *A <: B (При замене B на A работоспособность не нарушится)*
- *B <: A*
- *A и B не связаны*

Правило отношения между функциями

- Если A2 <: A1 и B1 <: B2 тогда (A1 => B1) <: (A2 => B2)
- Таким образом функции
  - *Контрвариантны в типах аргументов*
  - *Коварaитны в типе реузльтата*
- Описание функции

```
Trait Function1[-T,+U]{
    def apply (x:T):U
}
```

- ф

Правило отношения между функциями

- Если A2 <: A1 и B1 <: B2 тогда (A1 => B1) <: (A2 => B2)
- Таким образом функции
  - *Контрвариантны в типах аргументов*
  - *Ковараитны в типе реузльтата*
- Описание функции

```
Trait Function1[-T,+U]{
   def apply (x:T):U
}
```

Проверка вариативности

- Scala компилятор проверяет за нас отсутствие проблемных комбинаций при компиляции классов к модификаторами вариативности
  - *Ковариантные параметры типов могут появляться только в результатах метода*
  - *Контрвариантные параметры типов могут появляться только в параметрах метода*
  - *Инвариантные параметры могут быть везде*
- Проверить самостоятельно определение функции

Проверка вариативности

- Как мы видели в примере с массивом – проблема в изменении элемента
- Если вынести метод update отдельно, то получим

```
class Array[+T]{
   def update(x:T)
}
```

- То есть проблемная комбинация:
  - *Ковариантный параметр типа T*
  - *Который появляется в виде параметра метода*

Конкатенация списков

- ++ (concat)
- Необходимо доказать ассоциативность данной операции

(xs ++ ys) ++ zs = xs ++ (ys++zs)

xs ++ Nil = xs

Nil ++ xx = xs

- Подобный свойства можно доказать методов структурной индукции
- Натуральная индукция – для доказательства P(n)
  - *P(b) - база*
  - *If P(n) then P(n+1)*

Структурная индукция

- Для доказательства свойства P(xs)
  - *P(Nil) – база*
  - *Для списка xs и элемента x показать if P(xs) then P(xs :: x) – шаг индукции*
- Вспомним определение concat:

```
def concat[T](xs:List[T], ys:List[T]) : List[T] = xs match {
        case List() => ys
        case z::zs =>z:: concat(zs, ys)
}
```

- И его свойства:
  - *Nil ++ ys = ys*
  - *(x :: xs) ++ ys = x :: (xs ++ ys)*

Ассоциативность ++ - База

- Nil
- Для левой части
  - *(Nil ++ ys) ++ zs*
  - *= ys ++ zs* *- по 1му свойству*
- Для правой части
  - *Nil ++ (ys ++ zs)*
  - *ys ++ zs* *- по 1му свойству*

Ассоциативность ++ - База

- Nil
- Для левой части
  - *(Nil ++ ys) ++ zs*
  - = *ys ++ zs* - *по 1му свойству*
- Для правой части
  - *Nil ++ (ys ++ zs)*
  - *=ys ++ zs* - *по 1му свойству*

Ассоциативность ++ - Шаг

- x :: xs
- Для левой части ((x :: xs) ++ ys) ++ zs
  - *= (x :: (xs ++ ys)) ++ zs* - *по 2му свойству*
  - *= x :: ((xs ++ ys) ++ zs)* - *по 2му свойству*
  - *= x :: (xs ++ (ys ++ zs))* - *по изначальнйо гипотезе*
- Для правой части (x :: xs) ++ (ys ++ zs)
  - *= x :: (xs ++ (ys ++ zs))* - *по 2му свойству*

Вопрос

- xs ++ Nil = xs
- Сколько шагов индукции потребуется
  - *1*
  - *2*
  - *3*
  - *4*
  - *5*

Вопрос

- xs ++ Nil = xs
- Сколько шагов индукции потребуется
  - *1*
  - *2*
  - *3*
  - *4*
  - *5*

Самостоятельно

- Самостоятельно доказать следующие свойства
  - *xs.reverse.reverse = xs*
  - *(xs ++ ys) map f = (xs map f) ++ (ys map f)*

Категории

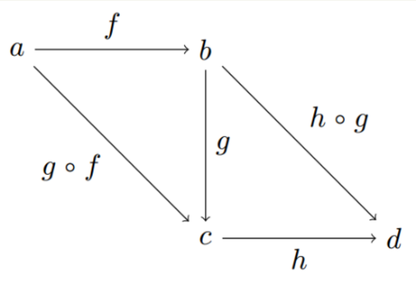
- Категория $\mathcal{C}$ состоит из:
  - *Объектов $a, b, c, \ldots$*
  - *Морфизмов (стрелок) $f, g, h, \ldots$*
  - *Для каждого морфизма f, задаются доменный и кодоменный объекты $a = \mathrm{dom}(f)$ и $b = \mathrm{cod}(f)$. Для записи $f : a \to b$.*
  - *Для каждого объекта a, задан идентифицирующий(тождественный) морфизм $\mathrm{id}_a : a \to a$.*
  - *Для каждой пары морфизм $f : a \to b$ и $g : b \to c$, существует композиция $g \circ f : a \to c$. То есть для каждой пары морфизм f и g с $\mathrm{cod}(f) = \mathrm{dom}(g)$, композиционный морфизм $g \circ f : \mathrm{dom}(f) \to \mathrm{cod}(g)$.*

Категории

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[dr, "f\,;g"'] & B \arrow[d, "g"] \\ & C \end{tikzcd}$$

Аксиомы категорий

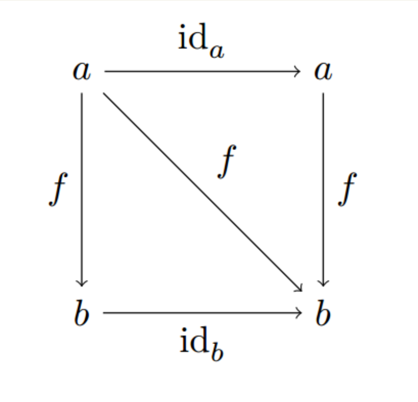
- Для всех морфизмов $f : a \to b$, $g : b \to c$, и $h : c \to d$, $h \circ (g \circ f) = h \circ g \circ f = (h \circ g) \circ f$, композиция морфизмов ассоциативна, что эквивалентно коммутативности:



- Для всех морфизм $f : a \to b$, $\mathrm{id}_b \circ f = f = f \circ \mathrm{id}_a$,

тождественный морфизмы тождественный для

композиции морфизмов, или



Спасибо за внимание!

- Планы на следующую лекцию
  - *Монады*