Данные не нужны

Что знал Алонзо Чёрч ещё 80 лет назад

Денис Буздалов

6 марта 2020
(13 ноября 2019)

Предположения докладчика

Предположения докладчика

Слушатели

- Воспринимают *густые* слайды

Предположения докладчика

Слушатели

- Воспринимают *густые* слайды

  с последовательно вываливающимися элементами

Предположения докладчика

Слушатели

- Воспринимают *густые* слайды

  с последовательно вываливающимися элементами
- Легко читают код со слайдов (синтаксис Haskell)

Предположения докладчика

Слушатели

- Воспринимают *густые* слайды

  с последовательно вываливающимися элементами
- Легко читают код со слайдов (синтаксис Haskell)
- Стремятся задать вопрос, когда непонятно

Предположения докладчика

- Типы-произведения

```
data Product a b = Product a b
```

Предположения докладчика

- Типы-произведения

```
data Product a b = Product a b
```

```
(a, b)
```

Предположения докладчика

- Типы-произведения

```
data Product a b = Product a b
```

```
(a, b)
```

- Типы-суммы

```
data Coproduct a b = A a | B b
```

Предположения докладчика

- Типы-произведения

```
data Product a b = Product a b
```

```
(a, b)
```

- Типы-суммы

```
data Coproduct a b = A a | B b
```

```
Either a b
```

Предположения докладчика

- Типы-произведения

```
data Product a b = Product a b
```

```
(a, b)
```

- Типы-суммы

```
data Coproduct a b = A a | B b
```

```
Either a b
```

- Algebraic data types

```
data X = A | B Int | C Int String
```

Предположения докладчика

- Типы-произведения

```
data Product a b = Product a b
```

```
(a, b)
```

- Типы-суммы

```
data Coproduct a b = A a | B b
```

```
Either a b
```

- Algebraic data types

```
data X = A | B Int | C Int String
```

- Полиморфные функции

```
f :: forall x. x → [x] → [x]
```

О чём доклад

- Зачем нам данные?
- Немножко об алгебре
- Снова о данных
- Снова об алгебре... алгебрах!
- Тайпклассы + полиморфизм = алгебра? `0_0`

О чём доклад

- Зачем нам данные?
- Немножко об алгебре
- Снова о данных
- Снова об алгебре... алгебрах!
- Тайпклассы + полиморфизм = алгебра? `0_0`

В начале будет весело, в конце сложно

О чём доклад

- Зачем нам данные?
- Немножко об алгебре
- Снова о данных
- Снова об алгебре... алгебрах!
- Тайпклассы + полиморфизм = алгебра? `0_0`

В начале будет весело, в конце сложно

И тут ещё больше лукавства

Для чего нужны данные?

```
parse :: Raw → Either Error Parsed
```

Для чего нужны данные?

```
parse :: Raw → Either Error Parsed
```

```
manageParsed :: Parsed → IO ()
```

Для чего нужны данные?

```haskell
parse :: Raw → Either Error Parsed

manageParsed :: Parsed → IO ()

manageRaw :: Raw → IO ()
manageRaw = m . parse where
  m (Right d) = manageParsed d
  m (Left e)  = error $ show e
```

Для чего нужны данные?

```
parse :: Raw → Either Error Parsed

manageParsed :: Parsed → IO ()

manageRaw :: Raw → IO ()
manageRaw = m . parse where
  m (Right d) = manageParsed d
  m (Left e)  = error $ show e
```

Альтернативно

```
manageRaw = either (error . show) manageParsed . parse
```

Для чего нужны данные?

```haskell
parse :: Raw → Either Error Parsed

manageParsed :: Parsed → IO ()

manageRaw :: Raw → IO ()
manageRaw = m . parse where
  m (Right d) = manageParsed d
  m (Left e)  = error $ show e
```

Альтернативно

```haskell
manageRaw = either (error . show) manageParsed . parse
```

Напоминание

```haskell
either :: (a → c) → (b → c) → Either a b → c
```

Для чего нужны данные?

```
split :: Whole → (Part1, Part2)
```

Для чего нужны данные?

```
split :: Whole → (Part1, Part2)
```

```
manageSplit :: Part1 → Part2 → IO ()
```

Для чего нужны данные?

```haskell
split :: Whole → (Part1, Part2)


manageSplit :: Part1 → Part2 → IO ()

manageWhole :: Whole → IO ()
manageWhole = m . split where
  m (p1, p2) = manageSplit p1 p2
```

Для чего нужны данные?

```haskell
split :: Whole → (Part1, Part2)

manageSplit :: Part1 → Part2 → IO ()

manageWhole :: Whole → IO ()
manageWhole = m . split where
  m (p1, p2) = manageSplit p1 p2
```

Альтернативно

```haskell
manageWhole = uncurry manageSplit . split
```

Для чего нужны данные?

```
split :: Whole → (Part1, Part2)

manageSplit :: Part1 → Part2 → IO ()

manageWhole :: Whole → IO ()
manageWhole = m . split where
  m (p1, p2) = manageSplit p1 p2
```

Альтернативно

```
manageWhole = uncurry manageSplit . split
```

Напоминание

```
uncurry :: (a → b → c) → (a, b) → c
```

Вспомним алгебру

- Символы
- Операции
- Отношения
- Свойства (законы)

Вспомним ~~алгебру~~ элементарную алгебру

- Символы: 0, 1, a, b, c, ...
- Операции: $a + b$, ab, a^b, ...
- Отношения: “$=$”
- Свойства (законы)

Вспомним ~~алгебру~~ элементарную алгебру

- Символы: 0, 1, a, b, c, ...
- Операции: $a + b$, ab, a^b, ...
- Отношения: “$=$”
- Свойства (законы):
  - $a + b = b + a$, $ab = ba$

Вспомним ~~алгебру~~ элементарную алгебру

- Символы: 0, 1, a, b, c, ...
- Операции: $a + b$, ab, a^b, ...
- Отношения: “$=$”
- Свойства (законы):
  - $a + b = b + a$, $ab = ba$
  - $a + (b + c) = (a + b) + c$
  - $a(bc) = (ab)c$

Вспомним ~~алгебру~~ элементарную алгебру

- Символы: 0, 1, a, b, c, ...
- Операции: $a + b$, ab, a^b, ...
- Отношения: “$=$”
- Свойства (законы):
  - $a + b = b + a$, $ab = ba$
  - $a + (b + c) = (a + b) + c$
  - $a(bc) = (ab)c$
  - $a(b + c) = ab + ac$

Вспомним ~~алгебру~~ элементарную алгебру

- Символы: 0, 1, a, b, c, ...
- Операции: $a + b$, ab, a^b, ...
- Отношения: “$=$”
- Свойства (законы):
  - $a + b = b + a$, $ab = ba$
  - $a + (b + c) = (a + b) + c$
  - $a(bc) = (ab)c$
  - $a(b + c) = ab + ac$
  - $1a = a$, $a + 0 = a$, $0a = 0$

Вспомним ~~алгебру~~ элементарную алгебру

- Символы: 0, 1, a, b, c, ...
- Операции: $a + b$, ab, a^b, ...
- Отношения: “$=$”
- Свойства (законы):
  - $a + b = b + a$, $ab = ba$
  - $a + (b + c) = (a + b) + c$
  - $a(bc) = (ab)c$
  - $a(b + c) = ab + ac$
  - $1a = a$, $a + 0 = a$, $0a = 0$
  - $a^0 = 1$, $a^1 = a$

Вспомним ~~алгебру~~ элементарную алгебру

- Символы: 0, 1, a, b, c, ...
- Операции: $a + b$, ab, a^b, ...
- Отношения: “$=$”
- Свойства (законы):
  - $a + b = b + a$, $ab = ba$
  - $a + (b + c) = (a + b) + c$
  - $a(bc) = (ab)c$
  - $a(b + c) = ab + ac$
  - $1a = a$, $a + 0 = a$, $0a = 0$
  - $a^0 = 1$, $a^1 = a$
  - $(ab)^c = a^c b^c$
  - $a^{b+c} = a^b a^c$
  - $a^{bc} = (a^b)^c$

Алгебра типов

- Символы: типы, kind `*`

  `Int` `[Int]` `Either String Int` ...

Алгебра типов

- Символы: типы, kind `*`
  `Int` `[Int]` `Either String Int` ...
- Операции: конструкторы типов
  - унарные, kind `* → *`
    `Maybe` `forall a. [a]` `Either String` ...

Алгебра типов

- Символы: типы, kind `*`
  `Int` `[Int]` `Either String Int` ...
- Операции: конструкторы типов
  - унарные, kind `* → *`
    `Maybe` `forall a. [a]` `Either String` ...
  - бинарные, kind `* → * → *`
    `Either` `(→)` `forall a b. (a, b)` ...

Алгебра типов

- Символы: типы, kind `*`
  `Int` `[Int]` `Either String Int` ...
- Операции: конструкторы типов
  - унарные, kind `* → *`
    `Maybe` `forall a. [a]` `Either String` ...
  - бинарные, kind `* → * → *`
    `Either` `(→)` `forall a b. (a, b)` ...
  - n-арные, kind `* → * → ... → *`

Алгебра типов

- Символы: типы, kind $*$
  `Int` `[Int]` `Either String Int` ...
- Операции: конструкторы типов
  - унарные, kind $* \to *$
    `Maybe` `forall a. [a]` `Either String` ...
  - бинарные, kind $* \to * \to *$
    `Either` `(->)` `forall a b. (a, b)` ...
  - n-арные, kind $* \to * \to \ldots \to *$
- Отношения: изоморфизм “$\rightleftharpoons$”

Алгебра типов

- Символы: типы, kind $*$
  `Int` `[Int]` `Either String Int` ...
- Операции: конструкторы типов
  - унарные, kind $* \to *$
    `Maybe` `forall a. [a]` `Either String` ...
  - бинарные, kind $* \to * \to *$
    `Either` `(->)` `forall a b. (a, b)` ...
  - n-арные, kind $* \to * \to \ldots \to *$
- Отношения: изоморфизм “$\rightleftharpoons$”

$$a \rightleftharpoons b \quad \equiv \quad \exists f\colon a \to b, g\colon b \to a \cdot f \circ g = id_b \wedge g \circ f = id_a$$

Алгебра типов

- Символы: типы, kind $*$
  `Int` `[Int]` `Either String Int` ...
- Операции: конструкторы типов
  - унарные, kind $* \to *$
    `Maybe` `forall a. [a]` `Either String` ...
  - бинарные, kind $* \to * \to *$
    `Either` `(→)` `forall a b. (a, b)` ...
  - n-арные, kind $* \to * \to \ldots \to *$
- Отношения: изоморфизм “$\rightleftharpoons$”

$$a \rightleftharpoons b \quad \equiv \quad \exists f\colon a \to b, g\colon b \to a \cdot f \circ g = id_b \wedge g \circ f = id_a$$

- Свойства: самое интересное

Алгебра типов: свойства

`Either A B` — сумма `A` и `B`, `(A, B)` — произведение

Алгебра типов: свойства

`Either A B` — сумма `A` и `B`, `(A, B)` — произведение

- Коммутативность
  - $a + b = b + a$

Алгебра типов: свойства

`Either A B` — сумма `A` и `B`, `(A, B)` — произведение

- Коммутативность
  - $a + b = b + a$

    `Either A B` $\rightleftharpoons$ `Either B A`

Алгебра типов: свойства

`Either A B` — сумма `A` и `B`, `(A, B)` — произведение

- Коммутативность
  - $a + b = b + a$

    `Either A B` $\rightleftharpoons$ `Either B A`
  - $ab = ba$

Алгебра типов: свойства

`Either A B` — сумма `A` и `B`, `(A, B)` — произведение

- Коммутативность
  - $a + b = b + a$

    `Either A B` $\rightleftharpoons$ `Either B A`
  - $ab = ba$

    `(A, B)` $\rightleftharpoons$ `(B, A)`

Алгебра типов: свойства

`Either A B` — сумма `A` и `B`, `(A, B)` — произведение

- Коммутативность
  - $a + b = b + a$

    `Either A B` $\rightleftharpoons$ `Either B A`
  - $ab = ba$

    `(A, B)` $\rightleftharpoons$ `(B, A)`
- Ассоциативность
  - $a + (b + c) = (a + b) + c$

Алгебра типов: свойства

`Either A B` — сумма `A` и `B`, `(A, B)` — произведение

- Коммутативность
  - $a + b = b + a$

    `Either A B` $\rightleftharpoons$ `Either B A`
  - $ab = ba$

    `(A, B)` $\rightleftharpoons$ `(B, A)`
- Ассоциативность
  - $a + (b + c) = (a + b) + c$

    `Either A (Either B C)` $\rightleftharpoons$ `Either (Either A B) C`

Алгебра типов: свойства

`Either A B` — сумма `A` и `B`, `(A, B)` — произведение

- Коммутативность
  - $a + b = b + a$

    `Either A B` $\rightleftharpoons$ `Either B A`
  - $ab = ba$

    `(A, B)` $\rightleftharpoons$ `(B, A)`
- Ассоциативность
  - $a + (b + c) = (a + b) + c$

    `Either A (Either B C)` $\rightleftharpoons$ `Either (Either A B) C`
  - $a(bc) = (ab)c$

Алгебра типов: свойства

`Either A B` — сумма `A` и `B`, `(A, B)` — произведение

- Коммутативность
  - $a + b = b + a$

    `Either A B` $\rightleftharpoons$ `Either B A`
  - $ab = ba$

    `(A, B)` $\rightleftharpoons$ `(B, A)`
- Ассоциативность
  - $a + (b + c) = (a + b) + c$

    `Either A (Either B C)` $\rightleftharpoons$ `Either (Either A B) C`
  - $a(bc) = (ab)c$

    `(A, (B, C))` $\rightleftharpoons$ `((A, B), C)`

Алгебра типов: свойства

- Дистрибутивность

Алгебра типов: свойства

- Дистрибутивность

  $a(b + c) = ab + ac$

Алгебра типов: свойства

- Дистрибутивность

  $a(b + c) = ab + ac$

  `(A, Either B C)` $\rightleftharpoons$ `Either (A, B) (A, C)`

Алгебра типов: свойства

```haskell
data () = () -- единица
```

Алгебра типов: свойства

```haskell
data () = () -- единица
data Void -- ноль
```

Алгебра типов: свойства

```haskell
data () = () -- единица
data Void -- ноль
```

- $a + 0 = a$

Алгебра типов: свойства

```haskell
data () = () -- единица
data Void -- ноль
```

- $a + 0 = a$

  `Either A Void` $\rightleftharpoons$ `A`

Алгебра типов: свойства

```haskell
data () = () -- единица
data Void -- ноль
```

- $a + 0 = a$

  `Either A Void` $\rightleftharpoons$ `A`
- $1a = a$

Алгебра типов: свойства

```haskell
data () = () -- единица
data Void -- ноль
```

- $a + 0 = a$

  `Either A Void` $\rightleftharpoons$ `A`
- $1a = a$

  `((), A)` $\rightleftharpoons$ `A`

Алгебра типов: свойства

```haskell
data () = () -- единица
data Void -- ноль
```

- $a + 0 = a$

  `Either A Void` $\rightleftharpoons$ `A`
- $1a = a$

  `((), A)` $\rightleftharpoons$ `A`
- $0a = 0$

Алгебра типов: свойства

```haskell
data () = () -- единица
data Void -- ноль
```

- $a + 0 = a$

  `Either A Void` $\rightleftharpoons$ `A`
- $1a = a$

  `((), A)` $\rightleftharpoons$ `A`
- $0a = 0$

  `(Void, A)` $\rightleftharpoons$ `Void`

Алгебра типов: свойства

- $1 + 1 = 2$

```
data Bool = True | False -- "двойка"
```

```
Either () () ⇌ Bool
```

Алгебра типов: свойства

- $1 + 1 = 2$

```
data Bool = True | False -- "двойка"
```

```
Either () () ≌ Bool
```

- $a + 1$

Алгебра типов: свойства

- $1 + 1 = 2$

```
data Bool = True | False -- "двойка"
```

```
Either () () ⇌ Bool
```

- $a + 1$

```
Either A ()
```

Алгебра типов: свойства

- $1 + 1 = 2$

```
data Bool = True | False -- "двойка"
```

```
Either () () ⇌ Bool
```

- $a + 1$

```
Either A () ⇌ ?
```

Алгебра типов: свойства

- $1 + 1 = 2$

```
data Bool = True | False -- "двойка"
```

```
Either () () ⇌ Bool
```

- $a + 1$

```
Either A () ⇌ Maybe A
```

Алгебра типов: свойства

- $1 + 1 = 2$

```
data Bool = True | False -- "двойка"
```

```
Either () () ⇌ Bool
```

- $a + 1$

```
Either A () ⇌ Maybe A
```

Вопрос:

```
Maybe Bool ⇌ ?
```

Алгебра типов: свойства

- $1 + 1 = 2$

```
data Bool = True | False -- "двойка"
```

```
Either () () ⇌ Bool
```

- $a + 1$

```
Either A () ⇌ Maybe A
```

Вопрос:

```
Maybe Bool ⇌ data Three = T1 | T2 | T3
```

Алгебра типов: свойства

Сколько существует функций

Алгебра типов: свойства

Сколько существует функций

- `Bool → Bool`

Алгебра типов: свойства

Сколько существует функций

- `Bool → Bool` 4

Алгебра типов: свойства

Сколько существует функций

- `Bool → Bool` 4
- `Bool → Three`

Алгебра типов: свойства

Сколько существует функций

- `Bool → Bool` 4
- `Bool → Three` 9

Алгебра типов: свойства

Сколько существует функций

- `Bool → Bool` 4
- `Bool → Three` 9
- `Three → Bool`

Алгебра типов: свойства

Сколько существует функций

- `Bool → Bool` 4
- `Bool → Three` 9
- `Three → Bool` 8

Алгебра типов: свойства

Функция `A` $\to$ `B` соответствует b^a

Алгебра типов: свойства

Функция `A` $\rightarrow$ `B` соответствует b^a

- $a^1 = a$

Алгебра типов: свойства

Функция `A → B` соответствует b^a

- $a^1 = a$

  `() → A` $\rightleftharpoons$ `A`

Алгебра типов: свойства

Функция `A → B` соответствует b^a

- $a^1 = a$

  `() → A` $\rightleftharpoons$ `A`

- $a^0 = 1$

Алгебра типов: свойства

Функция `A → B` соответствует b^a

- $a^1 = a$

  `() → A` $\rightleftharpoons$ `A`

- $a^0 = 1$

  `Void → A` $\rightleftharpoons$ `()`

Алгебра типов: свойства

- $(ab)^c = a^c b^c$

Алгебра типов: свойства

- $(ab)^c = a^c b^c$

  `C → (A, B)` $\rightleftharpoons$ `(C → A, C → B)`

Алгебра типов: свойства

- $(ab)^c = a^c b^c$

  `C → (A, B)` $\rightleftharpoons$ `(C → A, C → B)`

- $c^{ab} = c^{ba} = (c^b)^a$

Алгебра типов: свойства

- $(ab)^c = a^c b^c$

  `C → (A, B)` $\rightleftharpoons$ `(C → A, C → B)`

- $c^{ab} = c^{ba} = (c^b)^a$

  `(A, B) → C` $\rightleftharpoons$ `A → B → C`

Алгебра типов: свойства

- $(ab)^c = a^c b^c$

  ```
  C → (A, B) ⇌ (C → A, C → B)
  ```

- $c^{ab} = c^{ba} = (c^b)^a$

  ```
  (A, B) → C ⇌ A → B → C
  uncurry :: (a → b → c) → (a, b) → c
  ```

Алгебра типов: свойства

- $(ab)^c = a^c b^c$

  `C → (A, B)` $\rightleftharpoons$ `(C → A, C → B)`

- $c^{ab} = c^{ba} = (c^b)^a$

  `(A, B) → C` $\rightleftharpoons$ `A → B → C`

  ```
  uncurry :: (a → b → c) → (a, b) → c
  ```

- $c^{a+b} = c^a c^b$

Алгебра типов: свойства

- $(ab)^c = a^c b^c$

  `C → (A, B)` $\rightleftharpoons$ `(C → A, C → B)`

- $c^{ab} = c^{ba} = (c^b)^a$

  `(A, B) → C` $\rightleftharpoons$ `A → B → C`

  ```
  uncurry :: (a → b → c) → (a, b) → c
  ```

- $c^{a+b} = c^a c^b$

  `Either A B → C` $\rightleftharpoons$ `(A → C, B → C)`

Алгебра типов: свойства

- $(ab)^c = a^c b^c$

  `C → (A, B)` $\rightleftharpoons$ `(C → A, C → B)`

- $c^{ab} = c^{ba} = (c^b)^a$

  `(A, B) → C` $\rightleftharpoons$ `A → B → C`

  ```
  uncurry :: (a → b → c) → (a, b) → c
  ```

- $c^{a+b} = c^a c^b$

  `Either A B → C` $\rightleftharpoons$ `(A → C, B → C)`

  ```
  either :: (a → c) → (b → c) → Either a b → c
  ```

Алгебра типов: чего-то не хватает... ;-)

Алгебра типов: дифференцирование

Zipper (one-hole context)

Алгебра типов: дифференцирование

Zipper (one-hole context)

- $da = 1$

  `Der A` $\rightleftharpoons$ `()`

- $d(a^2) = 2a = a + a$

  `Der (A, A)` $\rightleftharpoons$ `(Bool, A)` $\rightleftharpoons$ `Either A A`

- $d(a + b) = da + db$

  `Der (Either A B)` $\rightleftharpoons$ `Either () ()` $\rightleftharpoons$ `Bool`

  `Der (Either a b)` $\rightleftharpoons$ `Either (Der a) (Der b)`

- $d(ab) = (da)b + a(db)$

  `Der (A, B)` $\rightleftharpoons$ `Either B A`

  `Der (a, b)` $\rightleftharpoons$ `Either (Der a, b) (a, Der b)`

Алгебра типов: интегрирование?

Алгебра типов: ”интегрирование”

- $\int f(x)dx$

Алгебра типов: ”интегрирование”

- $\int f(x)dx$

```
forall x. F x
```

Алгебра типов: ”интегрирование”

- $\int f(x)dx$

```
forall x. F x
```

- $\int (f(x) + g(x))dx = \int f(x)dx + \int g(x)dx$

Алгебра типов: ”интегрирование”

- $\int f(x)dx$

```
forall x. F x
```

- $\int (f(x) + g(x))dx = \int f(x)dx + \int g(x)dx$

```
forall x. Either (F x) (G x)  ⇌
          ⇌  Either (forall x. F x) (forall x. G x)
```

Алгебра типов: ”интегрирование”

- $\int f(x)dx$

  ```
  forall x. F x
  ```

- $\int (f(x) + g(x))dx = \int f(x)dx + \int g(x)dx$

  ```
  forall x. Either (F x) (G x) ⇌
            ⇌ Either (forall x. F x) (forall x. G x)
  ```

- $\int (c \cdot f(x))dx = c \cdot \int f(x)dx$

Алгебра типов: ”интегрирование”

- $\int f(x)dx$

  ```
  forall x. F x
  ```

- $\int (f(x) + g(x))dx = \int f(x)dx + \int g(x)dx$

  ```
  forall x. Either (F x) (G x) ⇌
            ⇌ Either (forall x. F x) (forall x. G x)
  ```

- $\int (c \cdot f(x))dx = c \cdot \int f(x)dx$

  ```
  forall x. (C, F x) ⇌ (C, forall x. F x)
  ```

Алгебра типов: к важному

Сколько существует функций

- $X \to X$

Алгебра типов: к важному

Сколько существует функций

- $X \to X$

  x^x

Алгебра типов: к важному

Сколько существует функций

- `X → X`

  x^x

- `(() → X) → X`

Алгебра типов: к важному

Сколько существует функций

- `X → X`

  x^x

- `(() → X) → X`

  x^{x^1}

Алгебра типов: к важному

Сколько существует функций

- `X → X`

  x^x

- `(() → X) → X`

  $x^{x^1} = x^x$

Алгебра типов: к важному

Сколько существует функций

- `X → X`

  x^x

- `(() → X) → X`

  $x^{x^1} = x^x$

- `(Void → X) → X`

Алгебра типов: к важному

Сколько существует функций

- `X → X`

  x^x

- `(() → X) → X`

  $x^{x^1} = x^x$

- `(Void → X) → X`

  x^{x^0}

Алгебра типов: к важному

Сколько существует функций

- `X → X`

  x^x

- `(() → X) → X`

  $x^{x^1} = x^x$

- `(Void → X) → X`

  $x^{x^0} = x$

Алгебра типов: к важному

Сколько существует функций

- `forall x. x → x`

Алгебра типов: к важному

Сколько существует функций

- `forall x. x → x`

  $\int x^x dx$

Алгебра типов: к важному

Сколько существует функций

- `forall x. x → x`

$$\int x^x dx = \int x^{x^1} dx$$

Алгебра типов: к важному

Сколько существует функций

- `forall x. x → x` $\rightleftharpoons$ `forall x. (() → x) → x`

  $\int x^x dx = \int x^{x^1} dx$

Алгебра типов: к важному

Сколько существует функций

- `forall x. x → x` $\rightleftharpoons$ `forall x. (() → x) → x`

  $\int x^x dx = \int x^{x^1} dx \sim 1$

Алгебра типов: к важному

Сколько существует функций

- `forall x. x → x ⇌ forall x. (() → x) → x ⇌ ()`

  $\int x^x dx = \int x^{x^1} dx \sim 1$

Алгебра типов: к важному

Сколько существует функций

- `forall x. x → x` ⇌ `forall x. (() → x) → x` ⇌ `()`

  $\int x^x dx = \int x^{x^1} dx \sim 1$

- `forall x. (Void → x) → x`

Алгебра типов: к важному

Сколько существует функций

- `forall x. x → x` ⇌ `forall x. (() → x) → x` ⇌ `()`

  $\int x^x dx = \int x^{x^1} dx \sim 1$

- `forall x. (Void → x) → x`

  $\int x^{x^0} dx$

Алгебра типов: к важному

Сколько существует функций

- `forall x. x → x` ⇌ `forall x. (() → x) → x` ⇌ `()`

  $\int x^x dx = \int x^{x^1} dx \sim 1$

- `forall x. (Void → x) → x`

  $\int x^{x^0} dx \sim 0$

Алгебра типов: к важному

Сколько существует функций

- `forall x. x → x` $\rightleftharpoons$ `forall x. (() → x) → x` $\rightleftharpoons$ `()`

  $\int x^x dx = \int x^{x^1} dx \sim 1$
- `forall x. (Void → x) → x` $\rightleftharpoons$ `Void`

  $\int x^{x^0} dx \sim 0$

Алгебра типов: к важному

Сколько существует функций

- `forall x. x → x` $\rightleftharpoons$ `forall x. (() → x) → x` $\rightleftharpoons$ `()`

  $\int x^x dx = \int x^{x^1} dx \sim 1$
- `forall x. (Void → x) → x` $\rightleftharpoons$ `Void`

  $\int x^{x^0} dx \sim 0$
- `forall x. (Bool → x) → x`

Алгебра типов: к важному

Сколько существует функций

- `forall x. x → x` $\rightleftharpoons$ `forall x. (() → x) → x` $\rightleftharpoons$ `()`

  $\int x^x dx = \int x^{x^1} dx \sim 1$
- `forall x. (Void → x) → x` $\rightleftharpoons$ `Void`

  $\int x^{x^0} dx \sim 0$
- `forall x. (Bool → x) → x`

  $\int x^{x^2} dx$

Алгебра типов: к важному

Сколько существует функций

- `forall x. x → x` $\rightleftharpoons$ `forall x. (() → x) → x` $\rightleftharpoons$ `()`

  $\int x^x dx = \int x^{x^1} dx \sim 1$
- `forall x. (Void → x) → x` $\rightleftharpoons$ `Void`

  $\int x^{x^0} dx \sim 0$
- `forall x. (Bool → x) → x`

  $\int x^{x^2} dx \sim 2$

Алгебра типов: к важному

Сколько существует функций

- `forall x. x → x` ⇌ `forall x. (() → x) → x` ⇌ `()`

  $\int x^x dx = \int x^{x^1} dx \sim 1$
- `forall x. (Void → x) → x` ⇌ `Void`

  $\int x^{x^0} dx \sim 0$
- `forall x. (Bool → x) → x` ⇌ `Bool`

  $\int x^{x^2} dx \sim 2$

Продолжения и кодирование Чёрча

$$\int x^{x^a} dx \sim a$$

```
forall x. (A → x) → x ⇌ A
```

Продолжения и кодирование Чёрча

$$\int x^{x^a} dx \sim a$$

```
forall x. (A → x) → x ⇌ A
```

```
data A = A1 | A2 B | A3 C D
```

$$a = (1 + b + cd) \sim$$

Продолжения и кодирование Чёрча

$$\int x^{x^a} dx \sim a$$

```
forall x. (A → x) → x ⇌ A
```

```
data A = A1 | A2 B | A3 C D
```

$$a = (1 + b + cd) \sim$$

$$\sim \int x^{x^{1+b+cd}} dx =$$

Продолжения и кодирование Чёрча

$$\int x^{x^a} dx \sim a$$

```
forall x. (A → x) → x ⇌ A
```

```
data A = A1 | A2 B | A3 C D
```

$$a = (1 + b + cd) \sim$$

$$\sim \int x^{x^{1+b+cd}} dx =$$

$$= \int x^{x x^b x^{cd}} dx = \int x^{x x^b x^{d^c}} dx$$

Продолжения и кодирование Чёрча

$$\int x^{x^a} dx \sim a$$

```
forall x. (A → x) → x ⇌ A
```

```
data A = A1 | A2 B | A3 C D
```

$$a = (1 + b + cd) \sim$$

$$\sim \int x^{x^{1+b+cd}} dx =$$

$$= \int x^{xx^bx^{cd}} dx = \int x^{xx^bx^{d^c}} dx$$

```
A ⇌ forall x. (x, B → x, C → D → x) → x
```

Продолжения и кодирование Чёрча

$$\int x^{x^a} dx \sim a$$

```
forall x. (A → x) → x ⇌ A
```

```
data A = A1 | A2 B | A3 C D
```

$$a = (1 + b + cd) \sim$$

$$\sim \int x^{x^{1+b+cd}} dx =$$

$$= \int x^{x x^b x^{cd}} dx = \int x^{x x^b x^{d^c}} dx$$

```
A ⇌ forall x. (x, B → x, C → D → x) → x
A ⇌ forall x. x → (B → x) → (C → D → x) → x
```

Классы типов спешат на помощь!

Классы типов спешат на помощь!

```haskell
data A = A1 | A2 B | A3 C D
```

Классы типов спешат на помощь!

```haskell
data A = A1 | A2 B | A3 C D

produce :: Int → A
produce x = if x < 0 then A1 else A2 $ toB x
```

Классы типов спешат на помощь!

```haskell
data A = A1 | A2 B | A3 C D


produce :: Int → A
produce x = if x < 0 then A1 else A2 $ toB x


onCase1 :: IO ()
onCase2 :: B → IO ()
onCase3 :: C → D → IO ()

manageA :: A → IO ()
manageA A1       = onCase1
manageA (A2 b)   = onCase2 b
manageA (A3 c d) = onCase3 c d
```

Классы типов спешат на помощь!

```
-- data A = A1 | A2 B | A3 C D
```

Классы типов спешат на помощь!

```
-- data A = A1 | A2 B | A3 C D
-- forall a. (a, B → a, C → D → a)
```

Классы типов спешат на помощь!

```
-- data A = A1 | A2 B | A3 C D

-- forall a. (a, B → a, C → D → a)

class A a where -- алгебра над `a`
  a1 :: a
  a2 :: B → a
  a3 :: C → D → a
```

Классы типов спешат на помощь!

```
-- data A = A1 | A2 B | A3 C D

-- forall a. (a, B → a, C → D → a)

class A a where -- алгебра над `a`
  a1 :: a
  a2 :: B → a
  a3 :: C → D → a


produce :: A a ⇒ Int → a
produce x = if x < 0 then a1 else a2 $ toB x
```

Классы типов спешат на помощь!

```haskell
-- data A = A1 | A2 B | A3 C D

-- forall a. (a, B → a, C → D → a)

class A a where -- алгебра над `a`
  a1 :: a
  a2 :: B → a
  a3 :: C → D → a


produce :: A a ⇒ Int → a
produce x = if x < 0 then a1 else a2 $ toB x


instance A (IO ()) where
  a1 = onCase1
  a2 = onCase2
  a3 = onCase3
```

Расширяемость

```
data A = A1 | A2 B | A3 C D
```

Расширяемость

```
data A = A1 | A2 B | A3 C D
```

Что, если хотим ещё?

```
data B = B1 | B2 E
```

Расширяемость

```haskell
data A = A1 | A2 B | A3 C D
```

Что, если хотим ещё?

```haskell
data B = B1 | B2 E
```

```haskell
manageBoth :: Either A B → X
manageBoth (Left A1)       = ...
manageBoth (Left (A2 b))   = ...
manageBoth (Left (A3 c d)) = ...
manageBoth (Right B1)      = ...
manageBoth (Right (B2 e))  = ...
```

Расширяемость

```
-- data A = A1 | A2 B | A3 C D
-- data B = B1 | B2 E
```

Расширяемость

```
-- data A = A1 | A2 B | A3 C D
-- data B = B1 | B2 E
```

$$(1 + b + cd) + (1 + e) = 1 + b + cd + 1 + e$$

Расширяемость

```
-- data A = A1 | A2 B | A3 C D
-- data B = B1 | B2 E
```

$$(1 + b + cd) + (1 + e) = 1 + b + cd + 1 + e$$

$$x^{(1+b+cd)+(1+e)} = xx^b x^{cd} xx^e$$

Расширяемость

```
-- data A = A1 | A2 B | A3 C D
-- data B = B1 | B2 E
```

$$(1 + b + cd) + (1 + e) = 1 + b + cd + 1 + e$$

$$x^{(1+b+cd)+(1+e)} = xx^b x^{cd} xx^e$$

```
class A a where ...
class B a where ...
```

Расширяемость

```
-- data A = A1 | A2 B | A3 C D
-- data B = B1 | B2 E
```

$$(1 + b + cd) + (1 + e) = 1 + b + cd + 1 + e$$

$$x^{(1+b+cd)+(1+e)} = xx^b x^{cd} xx^e$$

```
class A a where ...
class B a where ...

produceA    :: A a         ⇒ ... → a
produceBoth :: (A a, B a) ⇒ ... → a
```

Расширяемость

```
-- data A = A1 | A2 B | A3 C D
-- data B = B1 | B2 E
```

$$(1 + b + cd) + (1 + e) = 1 + b + cd + 1 + e$$

$$x^{(1+b+cd)+(1+e)} = xx^b x^{cd} xx^e$$

```
class A a where ...
class B a where ...

produceA    :: A a         ⇒ ... → a
produceBoth :: (A a, B a) ⇒ ... → a

instance A (IO ()) where ...
instance B (IO ()) where ...
```

Рекурсия (чуть-чуть)

Тип, для которого задаётся алгебра, может участвовать и в *аргументах* операций алгебры

```
class X a where
  f :: C → a
  g :: B → a → a
```

Это соответствует рекурсии в данных

Композиция

```
class Y a b where
  f :: C → a
  g :: B → a → b
```

Композиция

```
class Y a b where
  f :: C → a
  g :: B → a → b

h :: Y a b ⇒ C → B → b
h c b = g b $ f c
```

Композиция

```
class Y a b where
  f :: C → a
  g :: B → a → b

h :: Y a b ⇒ C → B → b
h c b = g b $ f c
```

А что, если...

```
instance Y (IO Int) (IO ()) where ...
```

Композиция с типами высших порядков

```
class Y m b where
  f :: C → m Int
  g :: B → Int → m b
```

Композиция с типами высших порядков

```haskell
class Y m b where
  f :: C → m Int
  g :: B → Int → m b

h :: (Y m b, Monad m) ⇒ C → B → b
h c b = g b =≪ f c
```

Композиция с типами высших порядков

```
class Y m b where
  f :: C → m Int
  g :: B → Int → m b

h :: (Y m b, Monad m) ⇒ C → B → b
h c b = g b =≪ f c
```

Если алгебра *предназначена* для монадической композиции

```
class Monad m ⇒ Y m b where ...
```

Но это накладывает свои ограничения!

Вернёмся к Either

```
class Errorable1 e a x where
  okay       :: a → x
  throwError :: e → x
```

Вернёмся к Either

```
class Errorable1 e a x where
  okay       :: a → x
  throwError :: e → x

class Errorable2 e a m where
  okay       :: a → m a
  throwError :: e → m a
```

Вернёмся к Either

```
class Errorable1 e a x where
  okay      :: a → x
  throwError :: e → x

class Errorable2 e a m where
  okay      :: a → m a
  throwError :: e → m a

class Errorable3 e m where
  okay      :: a → m a
  throwError :: e → m a
```

Вернёмся к Either

```
class Errorable1 e a x where
  okay       :: a → x
  throwError :: e → x

class Errorable2 e a m where
  okay       :: a → m a
  throwError :: e → m a

class Errorable3 e m where
  okay       :: a → m a
  throwError :: e → m a


class ... ⇒ Applicative m where
  pure :: a → m a
class Applicative m ⇒ Monad m where ...
class Monad m ⇒ MonadError e m where
  throwError :: e → m a
```

Вернёмся к Either

```
f :: ... → Either E A
```

Вернёмся к Either

```
f :: ... → Either E A
```

```
f :: MonadError E m ⇒ ... → m A
```

Вернёмся к Either

```
f :: ... → Either E A
Right a
```

```
f :: MonadError E m ⇒ ... → m A
pure a
```

Вернёмся к Either

```
f :: ... → Either E A
Right a
Left e
```

```
f :: MonadError E m ⇒ ... → m A
pure a
throwError e
```

Вернёмся к Either

```
f :: ... → Either E A
Right a
Left e
```

```
f :: MonadError E m ⇒ ... → m A
pure a
throwError e
```

Вернёмся к Either

```
f :: ... → Either E A
Right a
Left e
```

```
f :: MonadError E m ⇒ ... → m A
pure a
throwError e
```

```
ini2fin :: MonadError e m ⇒ Either e a → m a
ini2fin (Left e)  = throwError e
ini2fin (Right a) = pure a
```

Вернёмся к Either

```
f :: ... → Either E A
Right a
Left e
```

```
f :: MonadError E m ⇒ ... → m A
pure a
throwError e
```

```
ini2fin :: MonadError e m ⇒ Either e a → m a
ini2fin (Left e)  = throwError e
ini2fin (Right a) = pure a
```

```
fin :: MonadError String m ⇒ m Int
ini :: Either String Int
```

Вернёмся к Either

`f :: ... → Either E A`	`f :: MonadError E m ⇒ ... → m A`
`Right a`	`pure a`
`Left e`	`throwError e`

```
ini2fin :: MonadError e m ⇒ Either e a → m a
ini2fin (Left e)  = throwError e
ini2fin (Right a) = pure a
```

```
fin :: MonadError String m ⇒ m Int
ini :: Either String Int

ini = fin
fin = ini2fin ini
```

С состоянием и ошибками

```
helper :: MonadState St m ⇒ Int → m Intermediate

manage :: (MonadState St m, MonadError Err m)
       ⇒ Event → m Reaction
manage ev = do
  st ← get
  ...
  put $ f st ev
  ...
  inter ← helper $ num st
  ...
  if prop ev inter
    then pure $ g ev st
    else do
      put emptySt
      throwError $ Err1 "description"
```

Полубонус

Tagless-final и разные интерпретаторы кода

Пример: исходное состояние

```haskell
putStrLn :: String → IO ()
getLine  :: IO String

program :: IO ()
program = do
  putStrLn "What is your name?"
  name ← getLine
  putStrLn ("Hi, " ++ name)
```

Пример: исходное состояние

```haskell
putStrLn :: String → IO ()
getLine  :: IO String

program :: IO ()
program = do
  putStrLn "What is your name?"
  name ← getLine
  putStrLn ("Hi, " ++ name)
```

Как протестировать, что функция делает ровно то, что там надо?

Пример: выделили абстракцию

```haskell
class ConsoleIO m where
  putStrLn :: String → m ()
  getLine  :: m String
```

Пример: выделили абстракцию

```haskell
class ConsoleIO m where
  putStrLn :: String → m ()
  getLine  :: m String

program :: (Monad m, ConsoleIO m) ⇒ m ()
program = do
  putStrLn "What is your name?"
  name ← getLine
  putStrLn ("Hi, " ++ name)
```

Пример: выделили абстракцию

```haskell
class Monad m ⇒ ConsoleIO m where
  putStrLn :: String → m ()
  getLine  :: m String

program :: ConsoleIO m ⇒ m ()
program = do
  putStrLn "What is your name?"
  name ← getLine
  putStrLn ("Hi, " ++ name)
```

Пример: инстансы для обычного запуска

```haskell
instance ConsoleIO IO where
  putStrLn = Prelude.putStrLn
  getLine  = Prelude.getLine
```

Пример: инстансы для обычного запуска

```haskell
class MonadIO m where
  liftIO :: IO a → m a
```

```haskell
newtype RealConsoleT m a = RealConsoleT { runRealConsole :: m a }
  deriving (Functor, Applicative, Monad, MonadIO)

instance MonadIO m ⇒ ConsoleIO (RealConsoleT m) where
  putStrLn = liftIO . putStrLn
  getLine  = liftIO $ getLine
```

Пример: инстансы для необычного запуска

```haskell
newtype NoConsoleT m a = NoConsoleT { runNoConsole :: m a }
  deriving (Functor, Applicative, Monad, MonadIO)

instance Applicative m ⇒ ConsoleIO (NoConsoleT m) where
  putStrLn = const $ pure ()
  getLine  = pure ""
```

Пример: запуск программы

```
program :: ConsoleIO m ⇒ m ()

instance ConsoleIO IO where ...
instance MonadIO m ⇒ ConsoleIO (RealConsoleT m) where ...
instance Applicative m ⇒ ConsoleIO (NoConsoleT m) where ...
```

Пример: запуск программы

```
program :: ConsoleIO m ⇒ m ()

instance ConsoleIO IO where ...
instance MonadIO m ⇒ ConsoleIO (RealConsoleT m) where ...
instance Applicative m ⇒ ConsoleIO (NoConsoleT m) where ...



main = program
```

Пример: запуск программы

```
program :: ConsoleIO m ⇒ m ()

instance ConsoleIO IO where ...
instance MonadIO m ⇒ ConsoleIO (RealConsoleT m) where ...
instance Applicative m ⇒ ConsoleIO (NoConsoleT m) where ...



main = program

main = runRealConsole program
```

Пример: запуск программы

```
program :: ConsoleIO m ⇒ m ()

instance ConsoleIO IO where ...
instance MonadIO m ⇒ ConsoleIO (RealConsoleT m) where ...
instance Applicative m ⇒ ConsoleIO (NoConsoleT m) where ...



main = program

main = runRealConsole program

main = runNoConsole program
```

Пример: инстанс для тестирования

```
data ScenarioAction = ExpectPrinting String
                    | ExpectReading String

newtype TestingConsoleT m a = TestingConsoleT ...

instance (MonadError String m, MonadState [ScenarioAction] m)
      ⇒ ConsoleIO (TestingConsoleT m) where ...
```

Пример: инстанс для тестирования

```
data ScenarioAction = ExpectPrinting String
                    | ExpectReading String

newtype TestingConsoleT m a = TestingConsoleT ...

instance (MonadError String m, MonadState [ScenarioAction] m)
      ⇒ ConsoleIO (TestingConsoleT m) where ...



newtype SimpleTestRunningM = ExceptT String (State [ScenarioAction])
runTestingConsole :: [ScenarioAction]
                  → TestingConsoleT SimpleTestRunningM a
                  → Either String a
```

Пример: инстанс для тестирования

```
instance (MonadError String m, MonadState [ScenarioAction] m)
      ⇒ ConsoleIO (TestingConsoleT m) where
  putStrLn s = do
    sc ← get
    (curr, sc') ← case sc of
      []     → throwError "Scenario ended, but putStrLn"
      (x:xs) → pure (x, xs)
    put sc'
    case curr of
      ExpectPrinting exp →
        when (s /= exp) $ throwError "Wrong is printed"
      ExpectReader   _   → throwError "Reading is expected"

  getLine = ...
```

Пример: запуск для тестирования

```
scenario =
  [ ExpectPrintlng "What is your name?"
  , ExpectReading "Denis"
  , ExpectPrinting "Hi, Denis" ]

runTestingConsole scenario program -- gives Right ()
```

Пример: запуск для тестирования

```
scenario =
  [ ExpectPrintlng "What is your name?"
  , ExpectReading "Denis"
  , ExpectPrinting "Hi, Denis" ]

runTestingConsole scenario program -- gives Right ()
```

В hpec можно это использовать так:

```
spec = describe "Hello program" do
         it "asks and responds" do
           testRun `shouldBe` Right ()
  where
    testRun = runTestingConsole scenario program
```

В следующих сериях

- О композиции монад
- Трансформеры монад
- Другие системы эффектов

а также

- Кодирование Чёрча при рекурсии
- Рекурсивные схемы

Без стеснения вдохновлялся

- Chris Taylor, The Algebra of Algebraic Data Types (2012)
- Олег Нижников, Современное ФП с Tagless Final (2018)
- Edmund Noble, Data, and when not to use it (2018)
- Alexander Konovalov, Recursion schemes, algebras, finall tagless, data types (2019)
- John Hughes, Why functional programming matters (2016)
- Harold Carr, Refactoring Recursion (2019)
- и многими другими...

Элементы списка нажимабельны

Спасибо

Вопросы?