Пользовательские типы данных

Косарев Дмитрий a.k.a. Kakadu

матмех СПбГУ

13 сентября 2018 г.

Натуральные числа в стиле Пеано

Положим у нас есть “ноль” (ну или “один”) и есть “следующий”.

Натуральные числа в стиле Пеано

Положим у нас есть “ноль” (ну или “один”) и есть “следующий”.

```
data Nat = Zero | Succ Nat
```

Натуральные числа в стиле Пеано

Положим у нас есть “ноль” (ну или “один”) и есть “следующий”.

```
data Nat = Zero | Succ Nat
```

Простые упражнения: сложить или умножить пару чисел Пеано.

JSON

- true и false
- числа
- строки
- null
- массивы
- Объекты как набор пар “ключ-значение”

JSON

- true и false
- числа
- строки
- null
- массивы
- Объекты как набор пар “ключ-значение”

```
data JSON =
            JNull
          | JBool Bool
          | JNum   Float
          | JStr   String
          | JArray [JSON]
          | JObj   [(String, JSON)]
```

Простой пример: арифметика (1/2)

```
data Expr =
    EConst Int
  | EMul Expr Expr
  | EAdd Expr Expr
  deriving Show

eval :: Expr -> Int
```

Простой пример: арифметика (1/2)

```
data Expr =
    EConst Int
  | EMul Expr Expr
  | EAdd Expr Expr
  deriving Show

eval :: Expr -> Int

eval (EConst n) = n
eval (EAdd l r) = (eval l) + (eval r)
eval (EMul l r) = (eval l) * (eval r)
```

Простой пример: арифметика (2/2)

```
e1 :: Expr
e1 = EMul (EAdd (EConst 1) (EConst 2)) (EConst 3)
e2 = (1 + 2) * 3
```

Простой пример: арифметика (2/2)

```
e1 :: Expr
e1 = EMul (EAdd (EConst 1) (EConst 2)) (EConst 3)
e2 = (1 + 2) * 3

*Main> :l Arith
*Main> eval e1
9
*Main> e1
EMul (EAdd (EConst 1) (EConst 2)) (EConst 3)
```

Deep embedding vs. *shallow embedding*

Арифметика с переменными

```
data Expr =
    EConst Int
  | EMul Expr Expr
  | EAdd Expr Expr
  | EVar String
eval :: [(String,Int)] -> Expr -> Maybe Int
```

Арифметика с переменными

```
data Expr =
    EConst Int
  | EMul Expr Expr
  | EAdd Expr Expr
  | EVar String
eval :: [(String,Int)] -> Expr -> Maybe Int

eval _ (EConst n) = Just n
eval env (EAdd l r) =
  Just ((eval env l) + (eval env r))
eval env (EMul l r) =
  Just ((eval env l) * (eval env r))
eval env (EVar s) = lookup s env
```

Содержательный пример (1/3)

```
import Data.Word
data InetAddr = InetAddr Word8 Word8 Word8 Word8
data ConnectionState =
  Connecting  | Connected  | Disconnected
data ConnectionInfo = CInfo
  { state ::                    ConnectionState
  , server ::                   InetAddr
  , last_ping_time ::           Maybe Time
  , last_ping_id ::             Maybe Int
  , session_id ::               Maybe String
  , when_initiated ::           Maybe Time
  , when_disconnected ::        Maybe Time
  }
```

Содержательный пример (2/3)

```
data Connecting =
  Connecting { when_initiated:: Time }
data Connected  = Connected
  { last_ping  :: Maybe (Time,Int)
  , session_id :: String }
data Disconnected =
  Disconnected { when_disconnected :: Time }
data ConnectionState =
    SConnecting   Connecting
  | SConnected    Connected
  | SDisconnected Disconnected
```

Содержательный пример (3/3)

```
data ConnectionState =
    SConnecting   Connecting
  | SConnected    Connected
  | SDisconnected Disconnected
data ConnectionInfo = Cinfo
  { state ::  ConnectionState
  , server :: InetAddr }
```

Содержательный пример (3/3)

```
data ConnectionState =
    SConnecting   Connecting
  | SConnected    Connected
  | SDisconnected Disconnected
data ConnectionInfo = Cinfo
  { state ::  ConnectionState
  , server :: InetAddr }
```

Лозунг: “плохие” состояния (значения) должны быть непредставимы в типах.

Пример про почту (1/2)

```
data Contact = Contact
    { name :: Name
    , emailContactInfo :: EmailContactInfo
    , postalContactInfo :: PostalContactInfo }
```

Пример про почту (1/2)

```
data Contact = Contact
    { name :: Name
    , emailContactInfo :: EmailContactInfo
    , postalContactInfo :: PostalContactInfo }
```

Хочется, чтобы у контакта был *хотя бы один* адрес: либо электронной, либо физической почты.

Пример про почту (1/2)

```
data Contact = Contact
    { name :: Name
    , emailContactInfo :: EmailContactInfo
    , postalContactInfo :: PostalContactInfo }
```

Хочется, чтобы у контакта был *хотя бы один* адрес: либо электронной, либо физической почты. Что вы думаете о вот таком?

```
data Contact = Contact
   { name :: Name
   , emailContactInfo :: Maybe EmailContactInfo
   , postalContactInfo :: Maybe PostalContactInfo }
```

Пример про почту (2/2)

```
data ContactInfo =
    EmailOnly    EmailContactInfo
  | PostOnly     PostalContactInfo
  | EmailAndPost EmailContactInfo PostalContactInfo
data Contact = Contact
  { name        :: Name
  , contactInfo :: ContactInfo
  }
```

Если, посмотрев на тип, сразу понятно какие состояния корректные, а какие нет, то это считает хорошим дизайном.

Пример взят отсюда.

Пример про длины (1/2)

```
{-Measures.hs -}

module Measures (Miles(), Kilometers, miles
   , kilometers, addKilometers, addMiles) where
data Miles = Mile Int
data Kilometers = KM Int

miles = Mile
kilometers x = KM x

addMiles :: Miles -> Miles -> Miles
addMiles (Mile x) (Mile y) = Mile (x+y)

addKilometers :: Kilometers -> Kilometers ->
                 Kilometers
addKilometers (KM x) (KM y) = KM (x+y)
```

Пример про длины (2/2)

```
{-Measures.hs -}

import Measures
x = (miles 1) `addMiles` (miles 2)
```

Пример про длины (2/2)

```
{-Measures.hs -}

import Measures
x = (miles 1) `addMiles` (miles 2)

y :: Miles
y = 1 `addMiles` (miles 2)
{-
  Long error message which requires a notion of
  type class to be understandable
   |
8 | y = 1 `addMiles` (miles 2)
   |     ^
-}
```

Здесь запрещено случайно складывать числа с милями, а мили с километрами.

Домашнее упражнение про длины

Сейчас у нас отдельная функция сложения для миль и отдельная километров. Сделайте одну функцию, которая берет две “длины в одинаковой системе” и выдает “длину в такой же системе”.

Вам пригодятся знания:

- код выше можно переиспользовать немного поменяв;
- тип `Void` можно описать вручную, он не встроенный;
- типовые параметры могут использоваться в правой части, а могут и не использоваться;
- фильмография Луи де Фюнеса может подсказать какой термин из функционального программирования надо гуглить.