Лекция 3: типы данных

Функциональное программирование на Haskell

Алексей Романов

21 февраля 2018

МИЭТ

Типы-«перечисления»

- Тип `Bool` не встроен в язык, а определён в стандартной библиотеке:

  ```
  data Bool = False | True deriving (...)
  ```

- Ключевое слово `data` начинает определение.
- `Bool` — название типа.
- `False` и `True` называются конструкторами данных.
- Читаем как «У типа `Bool` есть ровно два значения: `False` и `True`.»
- О `deriving` позже.

Типы-«перечисления»

- Тип `Bool` не встроен в язык, а определён в стандартной библиотеке:

  ```
  data Bool = False | True deriving (...)
  ```

- Ключевое слово `data` начинает определение.
- `Bool` — название типа.
- `False` и `True` называются конструкторами данных.
- Читаем как «У типа `Bool` есть ровно два значения: `False` и `True`.»
- Так определяется любой тип с фиксированным перечнем значений (как `enum` в других языках):

  ```
  data Weekday = Monday | Tuesday | ...
  data FileMode = Read | Write | Append | ...
  ```

- Какие образцы у этих типов (кроме переменных и `_`)?

Типы-«перечисления»

- Тип `Bool` не встроен в язык, а определён в стандартной библиотеке:

  ```
  data Bool = False | True deriving (...)
  ```

- Ключевое слово `data` начинает определение.
- `Bool` — название типа.
- `False` и `True` называются конструкторами данных.
- Читаем как «У типа `Bool` есть ровно два значения: `False` и `True`.»
- Так определяется любой тип с фиксированным перечнем значений (как `enum` в других языках):

  ```
  data Weekday = Monday | Tuesday | ...
  data FileMode = Read | Write | Append | ...
  ```

- Какие образцы у этих типов (кроме переменных и `_`)?
- Каждое значение (конструктор) — образец.

Типы-«структуры»

- `data` также используется для аналогов структур: типов с несколькими полями. Например,

  ```
  data Point = Point Double Double
  ```

- Первое `Point` — название типа.
- Второе — конструктора.
- Когда конструктор один, названия обычно совпадают. Всегда по контексту можно определить, что из них имеется в виду.
- Образцы этого типа

Типы-«структуры»

- `data` также используется для аналогов структур: типов с несколькими полями. Например,

```
data Point = Point Double Double
```

- Первое `Point` — название типа.
- Второе — конструктора.
- Когда конструктор один, названия обычно совпадают. Всегда по контексту можно определить, что из них имеется в виду.
- Образцы этого типа это

```
Point образец_Double образец_Double
```

- Зададим значение типа `Point` и функцию на них:

```
Prelude> let origin = Point 0 0

Prelude> let xCoord (Point x _) = x
Prelude> xCoord origin
0.0
```

Типы-«структуры»

- Конструктор является функцией

```
Prelude> :t Point
```

Типы-«структуры»

- Конструктор является функцией

```
Prelude> :t Point
Point :: Double -> Double -> Point
```

- Мы можем дать полям конструктора имена. Это:
  - Документирует их смысл.
  - Определяет функции, возвращающие их значения.

```
Prelude> data Point = Point { xCoord ::
    Double, yCoord :: Double } deriving Show
Prelude> :t yCoord
yCoord2 :: Point -> Double
Prelude> yCoord (Point 0 (-1))
-1.0
```

- Особенно полезно, когда полей много, и только по типу не понять, какое где.
- Если успеем, вернёмся к записям в конце лекции.

▸ Синтаксис записей

Алгебраические типы: общий случай

- Может быть несколько конструкторов с полями:

```
data IpAddress = IPv4 String | IPv6 String
```

- Читаем «У типа `IpAddress` есть два *вида* значений: `IPv4` и `IPv6`.» Создание значения:

```
Prelude> let googleDns = IPv4 "8.8.8.8"
```

- Образцы: `IPv4 образец_String` и `IPv6 образец_String`.
- Пример функции:

```
isLocalhost (IPv4 "127.0.0.1") = True
isLocalhost (IPv6 "0:0:0:0:0:0:0:1") = True
isLocalhost _ = False
```

Алгебраические типы: общий случай

- Может быть несколько конструкторов с полями:

```
data IpAddress = IPv4 String | IPv6 String
```

- Читаем «У типа `IpAddress` есть два *вида* значений: `IPv4` и `IPv6`.» Создание значения:

```
Prelude> let googleDns = IPv4 "8.8.8.8"
```

- Образцы: `IPv4 образец_String` и `IPv6 образец_String`.
- Пример функции:

```
isLocalhost (IPv4 "127.0.0.1") = True
isLocalhost (IPv6 "0:0:0:0:0:0:0:1") = True
isLocalhost _ = False
```

- Ещё пример типа:

```
data ImageFormat = JPG | PNG | Other String
```

Алгебраические типы

- Несколько полей одного конструктора соответствуют операции декартова произведения в теории множеств.
- А несколько конструкторов?

Алгебраические типы

- Несколько полей одного конструктора соответствуют операции декартова произведения в теории множеств.
- А несколько конструкторов «помеченному объединению».
- Сравните помеченное объединение с непомеченным `union` в C/C++.
- И с `boost::variant` (`std::variant` в C++17).

Алгебраические типы

- Несколько полей одного конструктора соответствуют операции декартова произведения в теории множеств.
- В логике конъюнкции, а в алгебре произведению.
- А несколько конструкторов «помеченному объединению».
- В логике дизъюнкции, а в алгебре сумме.
- Сравните помеченное объединение с непомеченным `union` в C/C++.
- И с `boost::variant` (`std::variant` в C++17).
- Тип функций — импликация в логике и возведение в степень в алгебре

Записи

К определению записей

```
Prelude> let aPoint = Point 0 (-1)
Prelude> aPoint { xCoord = 1 }
```

Записи

◂ К определению записей

```
Prelude> let aPoint = Point 0 (-1)
Prelude> aPoint { xCoord = 1 }
Point {xCoord = 1.0, yCoord = -1.0}
Prelude> let Point {xCoord = x'} = it
Prelude> x'
```

Записи

К определению записей

```
Prelude> let aPoint = Point 0 (-1)
Prelude> aPoint { xCoord = 1 }
Point {xCoord = 1.0, yCoord = -1.0}
Prelude> let Point {xCoord = x'} = it
Prelude> x'
1.0
```

Записи

К определению записей

```
Prelude> let aPoint = Point 0 (-1)
Prelude> aPoint { xCoord = 1 }
Point {xCoord = 1.0, yCoord = -1.0}
Prelude> let Point {xCoord = x'} = it
Prelude> x'
1.0
Prelude> :set -XNamedFieldPuns -XRecordWildCards
Prelude> let f (Point { xCoord, yCoord }) = xCoord +
    yCoord
Prelude> let g (Point { .. }) = x + y
```

Оба последних образца — сокращения

```
Point { xCoord = xCoord, yCoord = yCoord }.
```

```
Prelude> Point { xCoord = 0 }
```

Записи

◂ К определению записей

```
Prelude> let aPoint = Point 0 (-1)
Prelude> aPoint { xCoord = 1 }
Point {xCoord = 1.0, yCoord = -1.0}
Prelude> let Point {xCoord = x'} = it
Prelude> x'
1.0
Prelude> :set -XNamedFieldPuns -XRecordWildCards
Prelude> let f (Point { xCoord, yCoord }) = xCoord +
    yCoord
Prelude> let g (Point { .. }) = x + y
```

Оба последних образца — сокращения

```
Point { xCoord = xCoord, yCoord = yCoord }.
```

```
Prelude> Point { xCoord = 0 }
```

Это компилируется только с предупреждением.