CS314. Функциональное программирование

Лекция 5а. Обобщённые алгебраические типы данных

В. Н. Брагилевский

Направление «Фундаментальная информатика и информационные технологии»
Институт математики, механики и компьютерных наук имени И. И. Воровича
Южный федеральный университет

23 сентября 2016 г.

Алгебраические типы данных

Типы-перечисления

```
data Suit = Spades | Clubs | Diamonds | Hearts
          deriving (Show, Eq, Ord)
```

Типы-контейнеры

```
data Person = Person String Int
            deriving (Show)
data Point = Point Double Double
           deriving (Show)
```

Общий случай

```
type Radius = Double
data Shape = Circle Point Radius | Rectangle Point Point
           deriving (Show)
```

Типы с параметрами

Стандартный тип Maybe a

```
data Maybe a = Nothing | Just a
```

Стандартный тип Either a b

```
data Either a b = Left a | Right b
```

Рекурсивный параметризованный тип двоичного дерева

```
data Tree a = EmptyTree | Node a (Tree a) (Tree a)
    deriving (Show)
```

- Конструкторы значений и конструкторы типов
- Типовые параметры конструкторов типов

Конструкторы значений и их типы

```
ghci> :t Circle
Circle :: Point -> Radius -> Shape
ghci> :t Circle (Point 2.5 3.1)
Circle (Point 2.5 3.1) :: Radius -> Shape
ghci> :t Circle (Point 2.5 3.1) 5.0
Circle (Point 2.5 3.1) 5.0 :: Shape
```

```
ghci> :t Just
Just :: a -> Maybe a
ghci> :t Just 'x'
Just 'x' :: Maybe Char
```

```
ghci> :t Node
Node :: a -> Tree a -> Tree a -> Tree a
ghci> :t Node True
Node True :: Tree Bool -> Tree Bool -> Tree Bool
```

Представление арифметических выражений

Тип

```
data IntExpr = I Int                    -- целочисленная константа
             | Add IntExpr IntExpr      -- сумма двух выражений
             | Mul IntExpr IntExpr      -- произведение выражений
```

Выражение типа IntExpr

```
(I 5 `Add` I 1) `Mul` I 7
```

Вычисление

```
eval :: IntExpr -> Int
eval (I n) = n
eval (Add e1 e2) = eval e1 + eval e2
eval (Mul e1 e2) = eval e1 * eval e2
```

- Тип IntExpr — простой пример EDSL (Embedded Domain Specific Language, встроенный предметно-ориентированный язык).

Содержание

Расширение представления выражений

План расширения

- Возможность представления логических значений.
- Операция проверки выражений на равенство (аргументы — целочисленные или логические выражения, результат — логическое выражение).

Идея

- Прежний тип: `IntExpr`
- Новый тип: `Expr a`

Проблема

- Каков тип eval (пока что `eval :: IntExpr -> Int`)?
  - Что является результатом вычисления, Bool или Int?
  - Плохая идея: `Either Int Bool` или `Maybe (Either Int Bool)`.

Объявление GADT (generalized algebraic data type)

```
{-# LANGUAGE GADTs #-}

data Expr a where
    I   :: Int  -> Expr Int
    B   :: Bool -> Expr Bool
    Add :: Expr Int -> Expr Int -> Expr Int
    Mul :: Expr Int -> Expr Int -> Expr Int
    Eq  :: Eq a => Expr a -> Expr a -> Expr Bool
```

- Часть конструкторов возвращают Expr Bool, остальные — Expr Int.

Для сравнения

```
{-# LANGUAGE GADTSyntax #-}
data Tree a where
  EmptyTree :: Tree a
  Node :: a -> Tree a -> Tree a -> Tree a
```

Использование GADT

```
eval :: Expr a -> a
eval (I n) = n
eval (B b) = b
eval (Add e1 e2) = eval e1 + eval e2
eval (Mul e1 e2) = eval e1 * eval e2
eval (Eq  e1 e2) = eval e1 == eval e2
```

- Типы использованных конструкторов определяют конкретный возвращаемый функцией тип.

```
ghci> :t eval (I 10)
eval (I 10) :: Int
ghci> :t eval (Eq undefined undefined)
eval (Eq undefined undefined) :: Bool
```

Содержание

Проблема: голова и хвост пустого списка

```
ghci> head []
*** Exception: Prelude.head: empty list
ghci> tail []
*** Exception: Prelude.tail: empty list
```

Вариант 1

```
{-#LANGUAGE EmptyDataDecls, GADTs  #-}

data NotSafe
data Safe

data SafeList a t where
     Nil :: SafeList a NotSafe
     Cons :: a -> SafeList a t -> SafeList a Safe

safeHead :: SafeList a Safe -> a
safeHead (Cons x _) = x
```

```
x = safeHead (Cons 1 Nil)
y = safeHead Nil -- ошибка проверки типов
```

Попытка реализации safeTail

```
safeTail :: SafeList a Safe -> SafeList a t
safeTail (Cons _ xs) = xs
```

- Ошибка компиляции:
  `xs :: SafeList a t`
  Результат должен иметь тип `SafeList a t`, но это разные `t`!
- Более того, `xs` может оказаться как `Safe`, так и `NotSafe`.
- Идея: почему бы не вести учёт уровней «безопасности» («этот список пять раз Safe»)?

Пример: безопасные списки

Вариант 2

```
{-#LANGUAGE EmptyDataDecls, GADTs  #-}

data NotSafe
data Safe t

data SafeList a t where
  Nil :: SafeList a NotSafe
  Cons :: a -> SafeList a t -> SafeList a (Safe t)

safeHead :: SafeList a (Safe t) -> a
safeHead (Cons x _) =  x

safeTail :: SafeList a (Safe t) -> SafeList a t
safeTail (Cons _ xs) = xs
```

```
xs = Cons 1 (Cons 2 (Cons 3 Nil))
x = safeHead (safeTail (safeTail xs))
```

```
ghci> :t xs
xs :: SafeList Integer (Safe (Safe (Safe NotSafe)))
ghci> x
3
```

```
ghci> safeHead (safeTail (safeTail (safeTail xs)))

<interactive>:54:40:
    Couldn't match type 'NotSafe' with 'Safe t0'
    Expected type: SafeList Integer (Safe (Safe (Safe (Safe t0))))
      Actual type: SafeList Integer (Safe (Safe (Safe NotSafe)))
    In the first argument of 'safeTail', namely 'xs'
    In the first argument of 'safeTail', namely '(safeTail xs)'
```

Как это работает?

```
{-#LANGUAGE EmptyDataDecls, GADTs  #-}

data NotSafe
data Safe t

data SafeList a t where
  Nil :: SafeList a NotSafe
  Cons :: a -> SafeList a t -> SafeList a (Safe t)
```

- `EmptyDataDecls`: типы `Safe t` и `NotSafe` имеют в точности одно возможное значение (bottom, $\bot$, `undefined`), их цель — помочь при проверке типов.
- `GADTs`: два конструктора списка создают значения разных типов (это списки, но один пустой, а второй нет, причём разница именно в типах).

GADT: выводы

- GADT обобщают алгебраические типы данных в том смысле, что позволяют указывать конкретные возвращаемые конструкторами значений типы. В ADT эти типы всегда совпадают с определяемым типом.
- GADT позволяют использовать систему типов для проверки типов выражений во встроенных языках (EDSL — embedded domain specific language).