Функциональное программирование с зависимыми типами на языке Idris

Лекция 4. Типы как сущности первого класса, функции на типах

В. Н. Брагилевский

23 ноября 2017 г.

Факультет компьютерных наук, НИУ «Высшая школа экономики»

Институт математики, механики и компьютерных наук
имени И. И. Воровича, Южный федеральный университет (Ростов-на-Дону)

Синонимы типов

```
import Data.Vect

tri : Vect 3 (Double, Double)
tri = [(0.0, 0.0), (3.0, 0.0), (0.0, 4.0)]

Position : Type
Position = (Double, Double)

tri' : Vect 3 Position
tri' = [(0.0, 0.0), (3.0, 0.0), (0.0, 4.0)]

Polygon : Nat -> Type
Polygon n = Vect n Position

tri'' : Polygon 3
tri'' = [(0.0, 0.0), (3.0, 0.0), (0.0, 4.0)]
```

Функции на типах

Функции the и cast

```
the : (a : Type) -> a -> a
cast : Cast from to => from -> to

summate : Int -> String -> Maybe (String, Int)
summate s y =
    case the Int (cast y) of
      0 => Nothing
      n => let s' = s + n
           in Just ("Sum="++cast s'++"\n", s')

main : IO ()
main = replWith 0 "> " summate
```

Задача: выполнение операций с учётом типа

Версия 1

```
defaultVal : t
toString : t -> String
fromMaybeTy : Maybe t -> t
```

Версия 2

```
defaultVal : (t : Type) -> t
toString : (t : Type) -> t -> String
fromMaybeTy : (t : Type) -> Maybe t -> t
```

Ограничение множества типов

```
data Ty = TyNat | TyBool | TyString

evalType : Ty -> Type
evalType TyNat = Nat
evalType TyBool = Bool
evalType TyString = String
```

Версия 3

```
defaultVal : (ty : Ty) -> evalType ty
toString : (ty : Ty) -> evalType ty -> String
fromMaybeTy : (ty : Ty) -> Maybe (evalType ty)
                        -> evalType ty
```

Вычисление значения по умолчанию

```
defaultVal : (ty : Ty) -> evalType ty
defaultVal TyNat = 0
defaultVal TyBool = False
defaultVal TyString = ""
```

```
idris> defaultVal TyBool
False : Bool
idris> defaultVal TyNat
0 : Nat
idris> defaultVal TyString
"" : String
```

Вычисление строкового представления

```
showNat : Nat -> String
showNat Z = "z"
showNat (S k) = "s" ++ showNat k

showBool : Bool -> String
showBool False = "f"
showBool True = "t"

toString : (ty : Ty) -> evalType ty -> String
toString TyNat x = showNat x
toString TyBool x = showBool x
toString TyString x = x
```

Извлечение значения из Maybe t

```
fromMaybeTy : (ty : Ty) -> Maybe (evalType ty)
                        -> evalType ty
fromMaybeTy ty Nothing = defaultVal ty
fromMaybeTy _ (Just x) = x
```

```
idris> fromMaybeTy TyBool Nothing
False : Bool
idris> fromMaybeTy TyNat (Just 5)
5 : Nat
```

Вычисления в типовых аннотациях

```
defaultVal' : (ty : Ty) ->
          case ty of
              TyNat => Nat
              TyBool => Bool
              TyString => String

fromMaybeTy' : (ty : Ty) ->
            let t = evalType ty
            in Maybe t -> t
```

Функции на типах

Функции с переменным числом аргументов

Пример: функция-сумматор аргументов

```
adder  0  10       = 10
adder  1  0  5     = 5
adder  2  0  4  6  = 10
```

Аргументы функции adder:

- количество дополнительных аргументов
- начальное значение
- дополнительный аргумент
- ...

Идея решения

Первый аргумент позволяет вычислить тип функции adder полностью:

```
adder 0 : Int -> Int
adder 1 : Int -> Int -> Int
adder 2 : Int -> Int -> Int -> Int
```

- Функция AdderType, вычисляющая тип функции по заданному числу дополнительных аргументов.
- Собственно функция adder этого типа.

Решение

```
AdderType : (numargs : Nat) -> Type -> Type
AdderType Z t = t
AdderType (S k) t = t -> AdderType k t

adder : Num nt => (numargs : Nat) -> nt
                  -> AdderType numargs nt
adder Z acc = acc
adder (S k) acc = \n => adder k (n + acc)
```

Функции на типах

Типобезопасная функция sprintf

Типобезопасная функция sprintf

Использование

```
sprintf "Hello!"                = "Hello!"
sprintf "Answer : %d"  42       = "Answer : 42"
sprintf "%s No %d" "Slide" 8 = "Slide No 8"
```

Типы

```
sprintf "Hello!"     : String
sprintf "Answer: %d" : Int -> String
sprintf "%s No %d"   : String -> Int -> String
```

Компоненты решения

- Тип данных для описания форматов
- Функция из строк в тип данных для форматов
- Функция на типах, вычисляющая тип `sprintf`

Тип данных для форматов

```
data Format = Number Format
            | Str Format
            | Lit String Format
            | End
```

`"%s = %d"` $\Longrightarrow$ `Str (Lit " = " (Number End))`

Вычисление формата по заданной строке

```
toFormat : (xs : List Char) -> Format
toFormat [] = End
toFormat ('%' :: 'd' :: chars) =
             Number (toFormat chars)
toFormat ('%' :: 's' :: chars) =
             Str (toFormat chars)
toFormat ('%' :: chars) =
             Lit "%" (toFormat chars)
toFormat (c :: chars) =
  case toFormat chars of
    Lit lit cs => Lit (strCons c lit) cs
    fmt => Lit (cast c) fmt
```

Вычисление типа sprintf

```
SPrintfType : Format -> Type
SPrintfType (Number fmt) =
                Int -> SPrintfType fmt
SPrintfType (Str fmt) =
                String -> SPrintfType fmt
SPrintfType (Lit str fmt) =
                SPrintfType fmt
SPrintfType End = String
```

Вспомогательная функция

```
sprintfFmt : (fmt : Format) -> (acc : String)
                            -> SPrintfType fmt
sprintfFmt (Number fmt) acc =
       \i => sprintfFmt fmt (acc ++ show i)
sprintfFmt (Str fmt) acc =
       \str => sprintfFmt fmt (acc ++ str)
sprintfFmt (Lit lit fmt) acc =
       sprintfFmt fmt (acc ++ lit)
sprintfFmt End acc = acc
```

Функция sprintf

```
sprintf : (fmt : String) ->
          SPrintfType (toFormat (unpack fmt))
sprintf fmt = sprintfFmt _ ""
```

Основная программа

```
main : IO ()
main = do
  s <- getLine
  putStrLn (sprintf "%s = %d" s 42)
```

Список литературы

Brady, Edwin (March, 2017). *Type-Driven Development with Idris*. Manning Publications.