6 Определение новых типов

Следует заметить, что идентификаторы обозначающие имена типов и конструкторов в Haskell должны начинаться с заглавной буквы, в отличие от имен переменных, которые должны называться с строчной буквы.

6.1 Синонимы типов

Также как и в Standard ML прозрачная привязка к типу заключается во введении синонима типа с помощью конструкции `type`.

```
type Interval = (Double, Double)
plusInterval :: Interval -> Interval -> Interval
plusInterval (x1,y1) (x2,y2) = (x1+x2, y1+y2)

Main> plusInterval (1, 3.4) (2.5, 6)
(3.5,9.4) :: (Double,Double)
```

6.2 Определение типов с конструкторами

Определим тип данных, `Money` по аналогии с типом `money`, пример которого рассматривался в Standard ML.

```
data Money = Nomoney | Coin  Integer | Note Integer
                  | Check String Integer


amount Money -> Integer
amount Nomoney = 0
amount (Coin pence) = pence
amount (Note pounds) = 100*pounds
amount (Check _ pence) = pence

Main> amount Nomoney
0 :: Integer
Main> amount (Coin 10)
10 :: Integer
Main> amount (Check "Bank of England" 10)
10 :: Integer
Main> amount (Note 10)
1000 :: Integer
```

Другой рассмотренный ранее пример

```
data BTree = Empty | Leaf| Node BTree BTree;

countLeaves :: BTree -> Integer
countLeaves Empty = 0
countLeaves Leaf = 1
countLeaves (Node treel tree2) =
    (countLeaves treel) + (countLeaves tree2)

t = Node (Node Leaf Leaf) (Node (Node Leaf Leaf) Leaf)

Main> countLeaves t
5 :: Integer
```

Еще один пример, но уже с использованием переменных типа.

```
data Newlist a = Null |  a :-> (Newlist a);

lenList :: Newlist a -> Integer
lenList Null = 0
lenList (_:->t) = 1 + (lenList t)

Main> lenList Null
0 :: Integer
Main> lenList l
4 :: Integer
Main> lenList (3.3:-> (5.6 :-> Null))
2 :: Integer
Main> lenList ("a":-> ("bCD" :-> Null))
2 :: Integer
```

6.3 Классы типов

Бывают функции, которые могут быть определены на аргументах разных типов; по сути, они описывают схожие понятия, но определены для значений разных типов. Например, функция сравнения на равенство, говорящая о том, что два значения одного типа a равны, имеет тип `a -> a -> Bool`, или функция печати выражения имеет тип `a -> String`. Здесь тип `a` является любым типом, для которого сравнение на равенство или печать (преобразование в строку) имеют смысл. Это понятие как раз и кодируется в классах типов.

Классы типов позволяют определять функции с одинаковым именем для разных типов.

У классов типов есть имена. Также как и имена типов и конструкторов, они начинаются с большой буквы. Так, предопределенный класс сравнений на равенство называется `Eq`, а класс печати выражений имеет имя `Show`. Эти классы имеют следующие определения.

Класс Eq:

```
class Eq a where
    (==) :: a -> a -> Bool
    (/=) :: a -> a -> Bool
```

Класс Show:

```
class Show a where
    show :: a -> String
```

За ключевым словом `class` следует имя класса, тип-параметр и ещё одно ключевое слово `where`. Далее с отступами пишутся имена определённых в классе значений. Значения класса называются методами.

В экземплярах классов типов даём конкретное наполнение для методов класса типов. Определение экземпляра пишется так же, как и определение класса типа, но вместо `class` пишется `instance`, вместо некоторого типа — конкретный тип, а вместо типов методов — их реализацию.

Переопределим наш тип `Interval` и сделаем его экземпляром классов `Eq` и `Show`

```
data Interval = I Double Double

instance Eq Interval where
    (==) (I x1 y1) (I x2 y2)  = (x1 == x2) && (y1 == y2)
    (/=) a b = not (a == b)

instance Show Interval where
    show (I x y) = "["++(show x)++","++show(y)++"]"
```

После такого определения интервалы можно сравнивать на равенство и отображать (в том числе и в интерпретаторе).

```
Main> I (-3.4) 7.1
[-3.4,7.1] :: Interval
Main> (I 2 3) == (I 7 4)
False :: Bool
Main> (I 2 3) == (I 2 3)
True :: Bool
```

Определение класса `Num` выглядит следующим образом

```
class (Eq a, Show a) => Num a where
    (+), (-), (*)  :: a -> a -> a
    negate         :: a -> a
    abs, signum    :: a -> a
    fromInteger    :: Integer -> a
```

Для того, чтобы сделать тип `Interval` экземпляром класса Num нужно определить все перечисленные процедуры. Можно это сделать следующим образом.

```
instance Num Interval where
    (+) (I x1 y1) (I x2 y2) = I (x1+x2) (y1+y2)
    (-) (I x1 y1) (I x2 y2) = I (x1-y2) (y1-x2)
    (*) (I x1 y1) (I x2 y2) =
                        let ac = x1*x2
                            ad = x1*y2
                            bc = y1*x2
                            bd = y1*y2
                        in I (min (min ac ad)(min bc bd))
                             (max (max ac ad)(max bc bd))
    negate (I x y) = I (negate y) (negate x)
    abs (I x y) | signum x == signum y =
                        let ax = abs x
                            ay = abs y
                        in I (min ax ay) (max ax ay)
                | otherwise = I 0 (max (abs x)(abs y))
    signum  (I x y) = I (signum x) (signum y)
    fromInteger x = I (fromInteger x) (fromInteger x)
```

Теперь все эти операции можно выполнять с участием интервалов и целых чисел.

```
Main> (I 3 5) * (I (-3) 4)
[-15.0,20.0] :: Interval
Main> abs (I (-3) 4)
[0.0,4.0] :: Interval
Main> (I 2 7) + 4
[6.0,11.0] :: Interval
Main> (I 2 7) + ((I (-1) 3) * -4)
[-10.0,11.0] :: Interval
Main> (I (-1) 3) - (I 1 3)
[-4.0,2.0] :: Interval
```