Функциональное программирование с зависимыми типами на языке Idris

Лекция 7. Idris как система доказательства теорем

В. Н. Брагилевский

29 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)

Отношение порядка

```
data LTE  : (n, m : Nat) -> Type where
  LTEZero : LTE Z    right
  LTESucc : LTE left right -> LTE (S left)
                                  (S right)

GTE : Nat -> Nat -> Type
GTE left right = LTE right left

LT : Nat -> Nat -> Type
LT left right = LTE (S left) right

GT : Nat -> Nat -> Type
GT left right = LT right left

isLTE : (m, n : Nat) -> Dec (LTE m n)
```

Пример доказательства свойства отношения порядка

$$\forall x, y \in \mathbb{N} \quad \neg(x \leqslant y) \longrightarrow (x > y)$$

```
-- if not(x <= y) then (x > y)
not_lte__gt : Not (x `LTE` y) -> x `GT` y
not_lte__gt {x = Z} {y} contra =
                     void (contra LTEZero)
not_lte__gt {x = (S k)} {y = Z} contra =
                            LTESucc LTEZero
not_lte__gt {x = (S k)} {y = (S j)} contra =
     LTESucc (not_lte__gt (\pf =>
                        contra (LTESucc pf)))
```

Пример: упорядоченность элементов вектора

```
data Sorted : (xs : Vect n Nat) -> Type where
 SortedEmpty : Sorted []
 SortedOne : (x : Nat) -> Sorted [x]
 SortedMany : (x : Nat) -> (y : Nat) ->
             Sorted (y :: zs) -> (x `LTE` y)
             -> Sorted (x :: y :: zs)

sortedVec12 : Sorted [1,2]
sortedVec12 = SortedMany 1 2 (SortedOne 2)
                             (LTESucc LTEZero)
sortedVec012 : Sorted [0,1,2]
sortedVec012 = SortedMany _ _ sortedVec12
                              LTEZero
```

Разрешимость свойства упорядоченности

```
isSorted : (xs : Vect n Nat) ->
             Dec (Sorted xs)
```

На пути к верифицированной функции сортировки

```
data Sorted : (xs : Vect n Nat) -> Type where
    ...

data Permuted : (xs : Vect n Nat) ->
                (ys : Vect n Nat) -> Type where
    ...

verifiedSort :
        (xs:Vect n Nat) ->
        (ys:Vect n Nat ** (Sorted ys,
                           Permuted xs ys))
```

Доказательство утверждений с использованием равенства

Доказательство утверждений с использованием равенства

Вспомогательные функции

Сохранение равенства при отображении компонентов

```
cong : {f : a -> b} -> x = y -> f x = f y
```

Симметричность равенства

```
sym : x = y -> y = x
```

Использование равенства в доказательстве свойств

```
replace : {P : a -> Type} -> x = y -> P x
                                   -> P y
```

Реализация функций cong, sym и replace

```
cong : {f : a -> b} -> x = y -> f x = f y
cong Refl = Refl

sym : x = y -> y = x
sym Refl = Refl

replace : {P : a -> Type} -> x = y -> P x
                                   -> P y
replace Refl pf = pf
```

Доказательство утверждений с использованием равенства

Коммутативность сложения в $\mathbb{N}$

Сложение натуральных чисел

```
plus : Nat -> Nat -> Nat
plus Z j = j
plus (S x) j = S (plus x j)
```

Коммутативность

```
plusComm : (m, n : Nat) ->
           plus m n = plus n m
```

```
%default total

plus_Z : (n : Nat) -> n = plus n Z
plus_Z Z = Refl
plus_Z (S k) = cong (plus_Z k)

plus_S : (n, k : Nat) ->
                S (plus n k) = plus n (S k)
plus_S Z k = Refl
plus_S (S j) k = cong (plus_S j k)

plusComm : (m, n : Nat) -> plus m n = plus n m
plusComm Z n = plus_Z n
plusComm (S k) n =
  rewrite plusComm k n in plus_S n k
```

Конструкция rewrite/in

`rewrite eq in res`

- Принцип действия: rewrite ищет левую часть eq в типе цели и *переписывает* её на правую часть, после чего возвращается `res`.
- Конструкция rewrite реализована как синтаксический сахар поверх replace.

Свойства plus в стандартной библиотеке

```
idris> :apropos plus
Prelude.Nat.plus : Nat -> Nat -> Nat
Add two natural numbers.

Prelude.Nat.plusOneSucc : (right : Nat)
    -> fromInteger 1 + right = S right

Prelude.Nat.plusSuccRightSucc : (left : Nat)
    -> (right : Nat) -> S (left + right)
                         = left + S right

Prelude.Nat.plusZeroRightNeutral : (left : Nat)
    -> left + fromInteger 0 = left
...
```

```
idris> :doc plusCommutative
Prelude.Nat.plusCommutative : (left : Nat) ->
    (right : Nat) -> left + right = right + left
```

Доказательство утверждений с использованием равенства

Циклический сдвиг вектора: доказательство корректности

Циклический сдвиг вектора

```
rotate : Vect n a -> Vect n a
rotate [] = []
rotate (x :: xs) = xs ++ [x]
```

Циклический сдвиг вектора: доказательство

```
import Data.Vect

rotate : Vect n a -> Vect n a
rotate [] = []
rotate (x :: xs) = rotateProof (xs ++ [x])
  where
    rotateProof : Vect (len + 1) a ->
                  Vect (S len) a
    rotateProof {len} xs =
          rewrite plusCommutative 1 len in xs
```

Доказательство утверждений с использованием равенства

Свидетели чётности

Определение чётности числа

```
data Parity = Even | Odd

parity : (n : Nat) -> Parity
parity Z = Even
parity (S Z) = Odd
parity (S (S k)) = parity k
```

Чётность/нечётность, выраженная в типе

```
data Parity : Nat -> Type where
    Even : Parity (n + n)
    Odd  : Parity (S (n + n))
```

```
idris> Even {n=5}
Even : Parity 10
idris> Odd {n=6}
Odd : Parity 13
```

```
parity : (n : Nat) -> Parity n
```

Поиск свидетеля чётности

```
parity : (n : Nat) -> Parity n
parity Z = Even {n = Z}
parity (S Z) = Odd {n = Z}
parity (S (S k)) =
  case parity k of
    Even {n} =>
      rewrite plusSuccRightSucc n n
      in Even {n = S n}
    Odd {n} =>
      rewrite plusSuccRightSucc n n
      in Odd {n = S n}
```

Преобразование числа в двоичный вид

```
data Digit = I | O

natToBin : Nat -> List Digit
```

```
natToBinH : Nat -> List Digit
natToBinH Z = []
natToBinH k =
  case parity k of
    Even {n} => O :: natToBinH n
    Odd  {n} => I :: natToBinH n

natToBin : Nat -> List Digit
natToBin Z = [O]
natToBin k = reverse (natToBinH k)
```

Две проблемы

1. Это преобразование не является верифицированным (тип результата не зависит от аргумента).
2. Функция natToBinH не распознаётся как тотальная:

```
idris> :total natToBinH
Main.natToBinH is possibly not total
due to recursive path:
    Main.natToBinH, Main.natToBinH
```

Верифицированная версия

```
data Binary : Nat -> Type where
   BEnd : Binary Z
   BO : Binary n -> Binary (n + n)
   BI : Binary n -> Binary (S (n + n))

natToBinV : (n:Nat) -> Binary n
natToBinV Z = BEnd
natToBinV k =
  case parity k of
    Even{n} => BO (natToBinV n)
    Odd {n} => BI (natToBinV n)
```

- Эта реализация *не распознаётся* как тотальная.

Список литературы

- Brady, Edwin (March, 2017). *Type-Driven Development with Idris*. Manning Publications.
- *The Idris Tutorial*. URL: http://docs.idris-lang.org/en/latest/tutorial/index.html.
- *Theorem Proving*. URL: http://docs.idris-lang.org/en/latest/proofs/index.html.