Программирование с зависимыми типами на языке Idris

Лекции 6–7. Выражение отношений между значениями и доказательство теорем

В. Н. Брагилевский

18 февраля 2017 г.

Computer Science клуб (Санкт-Петербург)

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

Содержание

Выражение равенства в типах

Коммутативность сложения

Пустой тип и разрешимость

Отношение принадлежности

Отношение порядка

Выражение равенства в типах


LIBERTE,
ÉGALITÉ,
FRATERNITÉ
OU LA MORT

$$A = A$$

Равенство натуральных чисел

```
data EqNat : (n1 : Nat) -> (n2 : Nat) -> Type where
  Same : (num : Nat) -> EqNat num num

Idris> the (EqNat 3 3) (Same 3)
Same 3 : EqNat 3 3
Idris> the (EqNat 3 3) (Same _)
Same 3 : EqNat 3 3
```

```
Idris> the (EqNat 3 4) (Same _)
(input):1:5:When checking argument value
            to function Prelude.Basics.the:
        Type mismatch between
                EqNat num num (Type of Same num)
        and
                EqNat 3 4 (Expected type)

        Specifically:
                Type mismatch between
                        0
                and
                        1
```

Проверка натуральных чисел на равенство

```
checkEqNat : (num1 : Nat) -> (num2 : Nat) ->
             Maybe (EqNat num1 num2)

checkEqNat Z Z = Just (Same 0)
checkEqNat Z (S k) = Nothing
checkEqNat (S k) Z = Nothing
checkEqNat (S k) (S j) =
     case checkEqNat k j of
       Nothing => Nothing
       Just (Same j) => Just (Same (S j))
```

```
Idris> checkEqNat 3 3
Just (Same 3) : Maybe (EqNat 3 3)
Idris> checkEqNat 3 4
Nothing : Maybe (EqNat 3 4)
```

Пример: собственная реализация exactLength

```
exactLength' : (len : Nat) -> (input : Vect m a) ->
                              Maybe (Vect len a)

exactLength' {m} len input =
  case checkEqNat len m of
    Nothing => Nothing
    (Just (Same m)) => Just input
```

Встроенное в Idris равенство

```
data (=) : a -> b -> Type where
  Refl : x = x
```

```
Idris> the (3 = 3) Refl
Refl : 3 = 3
Idris> the (2 + 2 = 4) Refl
Refl : 4 = 4
Idris> the ("xxx" = "xxx") Refl
Refl : "xxx" = "xxx"
Idris> the (True = True) Refl
Refl : True = True
```

```
Idris> the (1 = 0) Refl
(input):1:5:When checking argument value
            to function Prelude.Basics.the:
        Type mismatch between
                0 = 0 (Type of Refl)
        and
                1 = 0 (Expected type)

        Specifically:
                Type mismatch between
                        0
                and
                        1
```

Пример: вторая реализация checkEqNat

```
checkEqNat : (num1 : Nat) -> (num2 : Nat) ->
             Maybe (num1 = num2)

checkEqNat Z Z =Just Refl
checkEqNat Z (S k) = Nothing
checkEqNat (S k) Z = Nothing
checkEqNat (S k) (S j) =
  case checkEqNat k j of
    Nothing => Nothing
    Just eq => Just (cong eq)
```

Функция cong

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

Пример: циклический сдвиг вектора

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

Циклический сдвиг вектора и конструкция rewrite/in

```
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
```

- rewrite/in переписывает тип цели с помощью равенства
- plusCommutative — это функция из стандартной библиотеки

Коммутативность сложения

```
plus_commutes_Z : (n : Nat) -> plus Z n = plus n Z
plus_commutes_Z Z = Refl
plus_commutes_Z (S k) =
                rewrite plus_commutes_Z k in Refl

plus_commutes_S : (k : Nat) -> (n : Nat)
                  -> S (plus n k) = plus n (S k)
plus_commutes_S k Z = Refl
plus_commutes_S k (S j) =
                rewrite plus_commutes_S k j in Refl

plus_commutes : (m, n : Nat) -> m + n = n + m
plus_commutes Z n = plus_commutes_Z n
plus_commutes (S k) n =
                  rewrite plus_commutes k n in
                  plus_commutes_S k n
```

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
...
```

Функция sym

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

prop : (n, k : Nat) -> n + S k = S (n+k)
prop n k = sym (plusSuccRightSucc n k)
```

Пустой тип и разрешимость

Пустой тип и выражение отрицания

```
data Void : Type where
```

Высказывание A — истинно либо ложно	**Тип** A — населён либо нет
$\neg A$	$A \to \bot$

```
not2eq3 : (2 = 3) -> Void
f Refl impossible

Not : Type -> Type
Not a = a -> Void
```

Разрешимость (decidability)

Определение (теория алгоритмов)

Свойство называется разрешимым, если существует всегда завершающийся алгоритм, определяющий, выполняется оно или нет.

Определение (Idris)

```
data Dec : (prop : Type) -> Type where
  Yes : (prf : prop) -> Dec prop
  No : (contra : prop -> Void) -> Dec prop
```

Пример: разрешимость равенства натуральных чисел

```
checkEqNatWeak : (num1 : Nat) -> (num2 : Nat)
                 -> Maybe (num1 = num2)

checkEqNat : (num1 : Nat) -> (num2 : Nat)
             -> Dec (num1 = num2)
```

```
zeroNotSucc : (0 = S k) -> Void
zeroNotSucc Refl impossible

succNotZero : (S k = 0) -> Void
succNotZero Refl impossible

noRec : (contra : (k = j) -> Void) -> (S k = S j) -> Void
noRec contra Refl = contra Refl

checkEqNat : (num1 : Nat) -> (num2 : Nat) -> Dec (num1 = num2)
checkEqNat Z Z = Yes Refl
checkEqNat Z (S k) = No zeroNotSucc
checkEqNat (S k) Z = No succNotZero
checkEqNat (S k) (S j) =
           case checkEqNat k j of
             (Yes prf) => Yes (cong prf)
             (No contra) => No (noRec contra)
```

Интерфейс DecEq

```
interface DecEq ty where
  decEq : (val1 : ty) -> (val2 : ty)
          -> Dec (val1 = val2)
```

Имеются реализации для большинства стандартных типов

Пример: ещё одна реализация exactLength

```
exactLength' : (len : Nat) -> (input : Vect m a)
                  -> Maybe (Vect len a)
exactLength' {m} len input =
    case decEq m len of
      Yes Refl => Just input
      No contra => Nothing
```

Отношение принадлежности

Задача: удаление элемента вектора

Задача

Дан вектор и значение содержащегося в нём элемента. Удалить первое вхождение этого элемента.

```
removeElem : DecEq a => a -> Vect (S n) a -> Vect n a

removeElem' : DecEq a => a -> Vect (S n) a
              -> Maybe (Vect n a)

removeElem'' : DecEq a => a -> Vect (S n) a
               -> (p ** Vect p a)

removeElem''' : DecEq a => (v : a)
                -> (xs : Vect (S n) a)
                -> Elem v xs -> Vect n a
```

Отношение принадлежности элемента вектору

```
data Elem : a -> Vect k a -> Type where
  Here : Elem x (x :: xs)
  There : (later : Elem x xs) -> Elem x (y :: xs)

hasOne : Elem 1 [1,2,3]
hasOne = Here

hasFalse : Elem False [True, True, False]
hasFalse = There (There Here)

hasZero : Elem 0 [1,2,3]
hasZero = There (There (There ?hasZero_rhs2))
```

Реализация и использование removeElem

```
removeElem : DecEq a => (v : a) -> (xs : Vect (S n) a)
                -> Elem v xs -> Vect n a
removeElem v (v :: ys) Here = ys
removeElem {n = Z} _ _ (There _) impossible
removeElem {n = (S k)} v (y :: ys) (There later) =
                y :: removeElem v ys later
```

```
Idris> removeElem 3 [1,3,2] (There Here)
[1, 2] : Vect 2 Integer
```

Автоматический поиск доказательства

```
removeElem_auto : DecEq a => (v : a)
                  -> (xs : Vect (S n) a)
                  -> {auto prf : Elem v xs} -> Vect n a
removeElem_auto v xs {prf} = removeElem v xs prf

Idris> removeElem_auto 3 [1,3,2]
[1, 2] : Vect 2 Integer
```

Разрешимость отношения принадлежности

```
not_in_nil : Elem v [] -> Void
not_in_nil Here impossible
not_in_nil (There _) impossible

not_in_tail : (contra : (v = x) -> Void)
           -> (contra1 : Elem v xs -> Void)
           -> Elem v (x :: xs) -> Void
not_in_tail contra contra1 Here = contra Refl
not_in_tail contra contra1 (There later) =
                                  contra1 later
```

```
isElem : DecEq a => (v : a) -> (xs : Vect n a)
                 -> Dec (Elem v xs)
isElem v [] = No not_in_nil
isElem v (x :: xs) =
    case decEq v x of
      Yes Refl => Yes Here
      No contra =>
          case isElem v xs of
            Yes prf => Yes (There prf)
            No contra1 => No (not_in_tail
                                  contra contra1)
```

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 (m `LTE` n)
```

Пример доказательства свойства отношения порядка

```
-- 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)
```

Список литературы


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


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.