Функциональное программирование с зависимыми типами на языке Idris

Лекция 6. Выражение отношений на данных

В. Н. Брагилевский

27 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)

Проблема реализации функции `exactLength`

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

exactLength {m} len xs =
  if m == len
    then Just ?exactLength_rhs
    else Nothing
```

```
Main.exactLength_rhs
                    a : Type
                    m : Nat
                    len : Nat
                    xs : Vect m a
       ------------------------------------
        Main.exactLength_rhs : Vect len a
```

Выражение равенства в типах

$$A = A$$

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)
```

Реализация exactLength

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

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

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
```

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) =
        map cong (checkEqNat k j)
```

Функция cong (Prelude)

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

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
not2eq3 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

Ещё одна реализация exactLength

```
exactLength : (len : Nat) -> Vect m a
                  -> Maybe (Vect len a)
exactLength {m} len xs =
    case decEq m len of
      Yes Refl => Just xs
      No _ => 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)
```

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`.