Лекция 10

Ilya Yaroshevskiy

16 ноября

Содержание

1 Неравенство

Определение.

$$a \leq b \Leftrightarrow \exists x. a + x = b$$

Зависимая пара:

- Значение $(x, a + x = b)$
- Тип `\Sigma (x : Nat) (a + x = b)`

Пример. Хотим доказать: $5 \leq 12$, тогда зависимая пара: `(5, idp) : \Sigma (x : Nat) (5 + x = 12)`

Сделаем GADT для неравенства

Определение.

```
\data less-or-equal (a b : Nat)  \with
  | zero, _ => base
  | suc a', suc b' => next (p : less-or-equal a' b')
```

Хотим написать фнукцию, которая будем перестраивать `less-or-equal` в тип с квантором существования (зависимая пара)

```
\func f1 {a b : Nat}  {p1 : less-or-equal a b} : less-or-equal'
  \elim a, b, p1
    | 0, b, base => (b, idp)
    | suc a, suc b, next (pr 1) =>
      \let (pb, ppr) => f1 pr1 \in (pb, pmap suc ppr)
```

где `less-or-equal'` — $\exists x. a + x = b$

Аренд умеет сам делать 1-2 перехода ао определению, например `suc (x + a) = suc b` = `x + suc a = suc b`

```
\func f2 {a b : Nat} (p1 : less-or-equal' a b) : less-or-equal a b
  | {0}, _ => base
  -- по идее это absurd (transport fs p ()),
  -- но здесь аренд с сотоянии сам найти противоречие
  | {suc a}, {0}, (xm p) => contradiction
  | {suc a}, {suc b}, (x, p) => next (f2 {a} {b} (x, pmap minus1 p))
```

Определение.

```
\func plus-assoc {a b c : Nat}  : (a + b) + c = a + (b + c) \elim c
  | 0 => idp
  | suc c =>
    -- Можем так:
    -- pmap suc plus-assoc
    -- Но можем попробовать сделать замену
    rewrite plus-assoc idp
    -- На самом деле внутри происходит transport, rewrite угадавает эту лямбду
    -- transport (\lam x => (a + b) + suc c = suc x) plus-assoc idp
```

2 Классы

Вспомним определение группы: это $\langle R, +, e \ : \ R, ^{-1} \ : \ R \to R \rangle$, такие что:

- $e + x = x$
- $x + e = x$
- $x + x^{-1} = e$
- $x^{-1} + x = e$

```
\class Preorder R
 --snip--

\instance OrdNat : Preorder Nat
  -- Это зависимый тип, который нужно доказать
  | <= (a b : Nat) => TruncP (\Sigma (r : Nat) (r + a = b))
  | <=-reflexive => inP ((0, idp))
  | <=-transitive {x} {y} {z} =>
    -- Было искушение написать \lam (t1 : x <= y) ...
    \lam (t1 : TruncP (\Sigma (r : Nat)) (r + x = y))
         (t2 : TruncP (\Sigma (r : Nat)) (r + y = z)) =>
         \case t1, t2 \with {
           inP (d1, p1), inP (d2, p2) => inP (d1 + d2, {?})
         }
```

Примечание. Про `x` можем доказывать что:

- `x : Type`
- `x : Prop`

$\forall a \ b \ : \ x.a = b$. `TruncP` делает свой аргумент `Prop`'ом, то есть теперь все доказательства равны между собой