Лекция 5.
Интуиционистская теория типов
Язык Аренд

Интуиционистская теория типов

- Пер Мартин-Лёф, 1972 год (An Intuitionistic Theory of Types), 1982 год (Constructive mathematics and computer programming).
- Примерно соответствует расширенному варианту λ_C из лямбда-куба (зависимые типы, Π и Σ типы и т.п.).
- Явно введённые в теорию индуктивные типы (расширение алгебраических типов) и W-типы.
- Иерархия универсумов.
- Специальный тип для равенства.

Равенство

- Структурное равенство: побитовая одинаковость. Разрешимо, но малопригодно для жизни:

$$\frac{2}{5} \neq \frac{4}{10}$$

- Доказательное равенство: на основании аксиоматики. Неразрешимо.

Равенство в ИТТ

- Введём отношение идентичности $I(A, a, b)$:

$$\frac{\Gamma \vdash A \ \mathbf{type} \qquad \Gamma \vdash a : A \quad \Gamma \vdash b : A}{\Gamma \vdash I(A, a, b) \ \mathbf{type}}$$

- Теория 1972 года с интенсиональным равенством:

$$\frac{\Gamma \vdash A \ \mathbf{type} \qquad \Gamma \vdash a : A}{\Gamma \vdash \mathbf{refl}_A(a) : I(A, a, a)}$$

- Теория 1982 года — с доказательным равенством. Вводим отношение равенства $(=)$, для него принимаются аксиомы и правила вывода для равенства, и вводим «каноническое доказательство» $\mathbf{r}$:

$$\frac{\Gamma \vdash a =_A b}{\Gamma \vdash \mathbf{r} : I(A, a, b)} \qquad \frac{\Gamma \vdash c : I(A, a, b)}{\Gamma \vdash a =_A b} \qquad \frac{\Gamma \vdash c : I(A, a, b)}{\Gamma \vdash c = \mathbf{r} : I(A, a, b)}$$

  Наличие канонического доказательства позволяет выводить равенство доказательств идентичности.

Гомотопическая Т.Т. Изоморфизм Карри-Ховарда-Воеводского

Владимир Александрович Воеводский, 2006 год (A very short note on homotopy λ-calculus).

Логика	λ-исчисление	Топология
Высказывание	Тип	Пространство
Доказательство	Терм	Точка в пространстве
Предикат	Зависимый тип	Расслоение
Равенство	*Выделенный* зависимый тип	Пространство путей
Доказательство равенства	Элемент равенства	Путь между точками

Определение

Будем считать два значения a и b в пространстве X равными, если они связаны путём: непрерывной функцией $f : I \to X$, где $I = [0, 1]$, такой, что $f(0) = a$, $f(1) = b$.

Из определения равенства по Лейбницу следует, что если некоторый предикат истинен для левой части непрерывного пути, то он истинен и для правой части.

Равенство на примере натуральных чисел

Определение

$a, b \in \mathbb{N}_0$. Тогда $a \equiv b$, если существует непрерывная $f : I \to \mathbb{N}_0$, причём $f(0) = a$, $f(1) = b$

Лемма

$a \equiv b$ тогда и только тогда, когда $a = b$.

Доказательство.

Пусть $a = b$. Положим $f(x) = a$. Тогда $f(0) = a = b = f(1)$.
Пусть $a \neq b$. Предположим, есть путь f. Рассмотрим $A = f^{-1}(a)$ и $X = f^{-1}(\mathbb{N}_0 \setminus \{a\})$. По дискретности топологии $\{a\}$ и $\mathbb{N}_0 \setminus \{a\}$ открыты. По непрерывности тогда A и X открыты. Также оба непусты (очевидно, $0 \in A$, $1 \in X$), $A \cap X = \varnothing$ и $A \cup X = I$. Значит, I несвязно, что не так. Противоречие. $\square$

О языке Аренд

- Разработан в Jet Brains.
- Назван в честь Аренда Гейтинга (Arend Heyting, 9 мая 1898, Амстердам — 9 июля 1980, Лугано) — буква «H» из BHK-интерпретации.
- Язык сейчас исключительно для доказательств математических фактов: в отличие, например, от Coq — широко применяется для верификации софта, может использоваться для построения кода по доказательствам.
- Основан на Гомотопической теории типов (HoTT) — развитии интуиционистской теории типов Мартина-Лёфа.

Краткий перечень языковых конструкций

- Синтаксис: отчасти вдохновлён TeX-ом.
- Реализует конструкции из λC: зависимые типы, пи- и сигма-типы.

```
\func fe : \Pi (P : Nat -> \Type) ->
          (\Pi (a : Nat) -> (P a)) -> \Sigma (x : Nat) (P x) =>
   \lam P proof_forall => (0, proof_forall 0)
```

  «если утверждение P истинно при всех натуральных аргументах, то существует натуральный аргумент, при котором оно истинно»
- Индуктивные типы — обобщение алгебраических.
- Предикативная иерархия универсумов (базовые типы имеют уровень 0, тип уровня k имеет уровень $k+1$).
- Гомотопическая теория типов — формализуется с помощью унивалентной кубической теории типов. Равенство понимается как путь — в отличии от системы индуктивных типов в теории типов Мартина-Лёфа.

Индуктивные типы

- Алгебраические типы: параметры типа не влияют на выбор конструктора

```
type 'a list = Nil | Cons a ('a list)
```

- Обобщённые алгебраические типы: допустимые конструкторы зависят от параметров типа

```
type _ constant =
  | Int : int -> int constant
  | Float : float -> float constant

Int 5 : int constant
Float 3.141 : float constant
```

- Обобщаем дальше (зависимые типы в параметрах, сложные конструкторы): W-типы, индуктивные типы в разных вариантах.

Индуктивные типы в Аренде

Простой случай — похоже на алгебраические типы:

```
\data Nat
   | zero
   | suc Nat
```

Случай сложнее:

```
\data Fin (x : Nat) \elim x
  | suc n => { fzero | fsuc (Fin n) }

fzero : Fin 15          -- ок
fsuc (fzero) : Fin 1    -- не типизируется
```

Аренд интересен наличием высших индуктивных типов, простой пример:

```
\data Int
  | \coerce pos Nat
  | neg Nat \with { zero => pos zero }
```

Элиминаторы для индуктивных типов

- Знакомы с элиминаторами начиная с алгебраических типов:

$$\frac{\Gamma \vdash L : \varphi \vee \psi, \quad \Gamma \vdash f : \varphi \to \tau, \quad \Gamma \vdash g : \psi \to \tau}{\Gamma \vdash \mathrm{Case}\ L\ f\ g : \tau}$$

- Вспомним натуральные числа:

```
\data Nat
    | zero
    | suc Nat
```

Построим зависимый элиминатор:

```
\func Nat-elim (P : Nat -> \Type)
               (z : P zero)
               (s : \Pi (n : Nat) -> P n -> P (suc n))
               (x : Nat) : P x \elim x
  | zero => z
  | suc n => s n (Nat-elim P z s n)
```

Доказательство с помощью элиминаторов

```
\func plus-commutes-zero (x : Nat) : x Nat.+ 0 = 0 Nat.+ x \elim x
  | 0 => idp                                          -- P 0
  | suc x => pmap suc (plus-commutes-zero x)     -- P x -> P (suc x)

-- Но то же можно сделать с помощью Nat-elim:

\func Nat-elim (P : Nat -> \Type)
               (z : P zero)
               (s : \Pi (n : Nat) -> P n -> P (suc n))
               (x : Nat) : P x \elim x
  | zero => z
  | suc n => s n (Nat-elim P z s n)

\func plus-commutes-zero' (x : Nat) : x Nat.+ 0 = 0 Nat.+ x =>
  Nat-elim (\lam x => x Nat.+ 0 = 0 Nat.+ x)
           (idp)
           (\lam x prev => pmap suc prev) x
```

Свойства равенства

Теорема

$a = a$. Если $a = b$, то $b = a$. Если $a = b$ и $b = c$, то $a = c$.

Доказательство.

Положим $f(x) := a$.
Положим $g(x) := f(1 - x)$. Тогда $f(0) = b$, $f(1) = a$, композиция непрерывных — непрерывна.
Пусть $f_1 : a \rightsquigarrow b$ и $f_2 : b \rightsquigarrow c$. Тогда

$$g(x) := \begin{cases} f_1(2 \cdot x), & x < 0.5 \\ f_2(2 \cdot x - 1), & x \geqslant 0.5 \end{cases}$$

$\square$

Равенство в Аренде

Компьютер дискретен, потому мы имитируем топологическую конструкцию.

- Интервал `I` $:=$ [left..right] — есть доступ только к граничным точкам, внутренность недоступна.
- Путь — индуктивный тип:

```
\data Path (A : I -> \Type) (a : A left) (a’ : A right)
  | path (\Pi (i : I) -> A i)
```

  path f : Path $A\ a\ a'$ означает, что $f : a \rightsquigarrow a'$. Заметим, что компилятор дополнительно проверяет f left $\twoheadrightarrow_\beta a$ и f right $\twoheadrightarrow_\beta a'$ — неуказанная в определении встроенная семантика для `path`.
- Равенство $a = a'$ (примем $a, a' : A$) — сокращение:

```
\func \infix 1 = {A : \Type} (a a’ : A) => Path (\lam _ => A) a a’
```

  path $f : a = a'$ означает, что path f : Path $A\ a\ a'$, то есть найдётся $f : a \rightsquigarrow a'$.

Как работать с I

```
\func \infix 1 = {A : \Type} (a a' : A) => Path (\lam _ => A) a a'
```

Докажем что-нибудь про равенство.

- Рефлексивность:

  ```
  \func eq-reflexive {A : \Type} {a : A} : a = a => path (\lam _ => a)
  ```

- Для транзитивности нужна середина интервала. Для этого воспользуемся элиминатором для I, также системное определение:

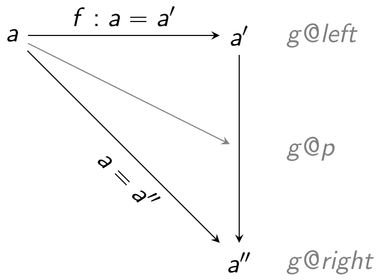
  ```
  \func coe (P : I -> \Type)
            (a : P left)
            (i : I) : P i \elim i
    | left => a
  ```

  «Если некоторый предикат P, индексированный интервалом, выполнен на левом крае интервала, то он выполнен в любой точке интервала»

Транзитивность равенства

```
\func eq-transitive {A : \Type} {a a’ a’’ : A}
                    (f : a = a’)
                    (g : a’ = a’’) : a = a’’ =>
  coe (\lam p => a = g @ p) f right
```

Проиллюстрируем картинкой:



Здесь $p \in \mathrm{I}$, и $g@p$ — некоторая точка по пути g из a' в a''.

Симметричность

Осталось третье свойство: $a = a' \to a' = a$.

```
\func eq-symmetric {A : \Type} {a a' : A} (p : a = a') : a' = a \elim p
  | idp => idp
```

Сопоставление с образцом для равенства возможно — его обитатель есть `idp`, специальный конструктор. Также встроенное в язык поведение.

Стандартные функции работы с равенством

- Если $a = a'$ и $B(a)$ истинен, то $B(a')$ истинен.

```
\func transport {A : \Type} (B : A -> \Type) {a a' : A}
                (p : a = a') (b : B a) : B a'
     => coe (\lam i => B (p @ i)) b right
```

  По сути тот же coe, но индексирует предикат вдоль пути вместо интервала:

```
\func eq-transitive {A : \Type} {a a' a'' : A}
                    (f : a = a')
                    (g : a' = a'') : a = a'' =>
  transport (\lam x => a = x) f right
```

- Если $a = a'$, то $f(a) = f(a')$:

```
\func pmap {A B : \Type} (f : A -> B) {a a' : A} (p : a = a') : f a = f a'
     => path (\lam i => f (p @ i))
```

Функциональная экстенциональность

Всюду совпадающие функции равны.

```
\func funExt {A : \Type} (B : A -> \Type) {f g : \Pi (a : A) -> B a}
    (p : \Pi (a : A) -> f a = g a) : f = g
    => path (\lam i => \lam a => p a @ i)
```

Доказать неравенство

- Ложь — это `Empty` (тип без значений).

```
\data Empty
```

- $0 \neq 1$ — это $0 = 1 \to \bot$.
- Построим функцию `T : Nat -> \Type`, которая для 0 возвращает обитаемый тип, а для других чисел — необитаемый.

```
\func T (x : Nat) : \Type \elim x
| Z => \Sigma
| _ => Empty
```

- T — это предикат, раз можем поставить вопрос, истинен (обитаем) ли $T\ 0$. Поэтому он сохраняет истинность для равных объектов (принцип Лейбница).
- Заметим, что `T 0` истинен (обитаем), так как `() : \Sigma`. Значит, мы найдём значение и для `T 1` (для `Empty`), при условии, что $0 = 1$.

```
zero-ne-1 (p : 0 = 1) : Empty => transport T p ()
```