Теория типов

Михайлов Максим

29 ноября 2021 г.

Оглавление

Лекция 1

7 сентября

1 Лямбда-исчисление

То, чем мы будем заниматься, можно назвать прикладной матлогикой.

В рамках курса матлогики мы рукомахательно рассмотрели изоморфизм Карри-Ховарда, в этом курсе мы его формализуем. Мы затронем систему типов Хиндли-Милнера *(Haskell)* и язык Arend, основанный на гомотопической теории типов.

1.1 Определение

В 20-30х годах XX века Алонзо Чёрчем была создана альтернатива теории множеств как основе математики — лямбда-исчисление. Основная идея — выбросить из языка все, кроме вызова функций.

В лямбда исчислении есть три конструкции:

- Функция *(абстракция)*: $(\lambda x.A)$
- Применение функции *(аппликация)*: $(A\ B)$
- Переменная *(атом)*: x

Большими буквами начала латинского алфавита мы будем обозначать термы, малыми буквам конца — переменные. λ жадная, как $\forall$ и $\exists$ в исчислении предикатов. Аппликация идёт слева направо, т.е. $\lambda p.p\ F\ T = \lambda p.((p\ F)\ T)$

Вычисление происходит с помощью β-редукции, его мы определим позже, общее понимание у нас есть из вводной лекции функционального программирования.

1.2 Булево исчисление

Определим булево исчисление в λ-исчислении:

- $T \coloneqq \lambda x.\lambda y.x$ — истина
- $F \coloneqq \lambda x.\lambda y.y$ — ложь
- $\text{Not} \coloneqq \lambda p.p\ F\ T$

$$\begin{aligned} \text{Not}\ F &\to_\beta \\ ((\lambda x.\lambda y.y)\ F)\ T &\to_\beta \\ (\lambda y.y)\ T &\to_\beta T \end{aligned}$$

- $\text{And} \coloneqq \lambda a.\lambda b.a\ b\ F$

And берёт свой второй аргумент, если первый аргумент истина и ложь иначе.

And использует идею **карринга** — функция от 2 аргументов есть функция от первого аргумента, возвращающая другую функцию от второго аргумента.[1] Например, в выражении “$((+)\ 2)\ 3$” $((+)\ 2)$ это функция, которая прибавляет к своему аргументу 2.

1.3 Числа

Числа в лямбда-исчислении кодируются **нумералами Чёрча**. Это только один из способов кодировки, есть и другие. Общая идея — число n применяет данную функцию к данному аргументу n раз.

- $0 = \lambda f.\lambda x.x$
- $1 = \lambda f.\lambda x.f\ x$
- $3 = \lambda f.\lambda x.f\ (f\ (f\ x))$
- $\overline{n+1} = \lambda f.\lambda x.f(\overline{n}\ f\ x)$
- $(+1) = \lambda n.\lambda f.\lambda x.n\ f\ (f\ x)$ — функция инкремента.
- $(+) = \lambda a.\lambda b.b\ ((+)\ \overline{1})\ a$: b раз прибавляет единицу к a.
- $(\cdot) = \lambda a.\lambda b.a\ ((+)\ b)\ \overline{0}$: a раз прибавляет b к 0.

Ходят легенды, что Клини изобрел декремент у зубного врача под действием наркоза. Существует много способов определить декремент различных степеней упоротости.

Рассмотрим декремент, основанный на следующей идее: пусть есть упорядоченная пара $\langle a, b\rangle$ и функция $(*) : \langle a, b\rangle \mapsto \langle b, b+1\rangle$. Тогда применив $(*)$ n раз к $\langle 0, 0\rangle$ и взяв первый элемент, возьмём первый элемент пары.

[1] Аналогично для n аргументов.

Упорядоченная пара определяется следующим способом:

$$\text{MkPair} = \lambda a.\lambda b.(\lambda p.p\ a\ b)$$

Можно потрогать эмулятор лямбда-исчисления `lci`, будет полезно для домашних заданий.

1.4 Типизированное лямбда-исчисление

Лямбда-исчисление для нас будет просто языком программирования. Для начала мы его типизируем, потому что нетипизированное лямбда-исчисление противоречиво.

Пусть у каждого выражения A есть тип τ, что обозначается $A : \tau$. Также используется некоторый контекст с переменными и их типами, обозначаемый M. Все вместе это записывается как $M \vdash A : \tau$, что напоминает исчисление предикатов.

1.5 Y-комбинатор и противоречивость нетипизированного λ-исчисления

Мы хотим, чтобы $\to_\beta$ сохраняло значения, т.к. иначе мы вообще не можем говорить о равенстве термов.

Определение. $Y \coloneqq \lambda f.(\lambda x.f\ (x\ x))(\lambda x.f\ (x\ x))$ — Y-комбинатор, для него верно $Yf \approx f(Yf)$. Такое свойство называется “быть **комбинатором неподвижной точки**”, т.е. он находит неподвижную точку функции: A такое, что $f(A) = A$.

Пусть мы добавили бинарную операцию $(\supset)$ — импликацию с некоторыми аксиомами. Оказывается, что доказуемо любое A. Мы это докажем на последующих лекциях.

Y-комбинатор полезен тем, что позволяет реализовывать рекурсию.

Пример. Запишем факториал в неформальном виде:

$$\text{Fact} = \lambda n.\text{If (IsZero } n)\ \overline{1}\ (\text{Fact } (n-1) \cdot n)$$

На самом деле Fact есть неподвижная точка функции

$$\lambda f.\lambda n.\text{If (IsZero } n)\ \overline{1}\ (f\ (n-1) \cdot n)$$

по определению неподвижной точки функции. Тогда Fact это

$$Y(\lambda f.\lambda n.\text{If (IsZero } n)\ \overline{1}\ (f\ (n-1) \cdot n))$$

У нас появляется проблема: есть выражения, которым мы не можем приписать значение, например

$$Y(\lambda f.\lambda x.f\ (\text{Not } x))$$

Эта проблема происходит из-за того, что наш язык слишком мощный — мы написали решатель любых уравнений, даже тех, у которых нет решения. Логичный выход из этой ситуации — запретить то, из-за чего у нас возникают проблемы. Как запретить Y? Оказывается, это позволяют сделать типы — они будут делить выражения на добропорядочные и недобропорядочные.

Лекция 2

14 сентября

2 Формализация λ-исчисления

Определение. **Пред-λ-терм** определяется индуктивно как одно из:

1. x — переменная
2. $(L\ L)$ — применение
3. $(\lambda x.L)$ — абстракция

Почему пред-λ-терм? Мы не хотим различать $\lambda x.x$ и $\lambda y.y$.

Определение. **α-эквивалентность** — обозначается $A =_\alpha B$ и выполняется, если[1]:

1. $A \equiv x, B \equiv x$ — одна и та же переменная
2. $A \equiv P\ Q, B \equiv R\ S, P =_\alpha R, Q =_\alpha S$
3. $A \equiv \lambda x.P, B \equiv \lambda y.Q$ и существует t — новая переменная, такая что $P[x \coloneqq t] =_\alpha Q[y \coloneqq t]$

Определение. Свобода для подстановки: $A[x \coloneqq B]$, никакое свободное вхождение переменной в B не станет связанным.

Определение (λ-терм). Множество всех λ-термов это $\Lambda/_{=_\alpha}$

Определение (β-редекс). Выражение вида $(\lambda x.A)\ B$

Определение (β-редукция). Обозначается $A \to_\beta B$ и выполняется, если выполняется одно из:

1. $A \equiv P\ Q, B \equiv R\ S$ и либо $P \to_\beta R$ и $Q =_\alpha S$, либо $P =_\alpha R$ и $Q =_\alpha S$.

[1] И только если.

2. $A \equiv \lambda x.P, B \equiv \lambda x.Q$ и $P \to_\beta Q$
3. $A \equiv (\lambda x.P)\ Q, B \equiv P[x \coloneqq Q]$ и Q свободно для подстановки.

Определение. Придуман Моисеем Шейнфинкелем.

$I \coloneqq \lambda x.x$ — Identität[2]

Определение.

- $K = \lambda x.\lambda y.x$
- $\Omega = \omega\ \omega$
- $\omega = \lambda x.x\ x$

Пример.

$$\begin{tikzcd}
KI\Omega \arrow[loop above] \arrow[r, "\beta"] \arrow[d, "\beta"] & ((\lambda x.\lambda y.x)I)\Omega \arrow[loop above] \\
(\lambda y.I)\Omega \arrow[loop left] \arrow[d, "\beta"] & \\
I &
\end{tikzcd}$$

Определение. R обладает **ромбовидным свойством** *(diamond)*, если для любых a, b, c, таких что:

1. aRb, aRc
2. $b \neq c$

существует d: bRd и cRd.

$$\begin{tikzcd}
& A \arrow[dl] \arrow[dr] & \\
B \arrow[dr] & & C \arrow[dl] \\
& D &
\end{tikzcd}$$

Пример. $>$ на $\mathbb{Z}$ не ромбовидно: для $a = 3, b = 2, c = 1$ выполнено условие, но $\nexists d$.

$>$ на $\mathbb{R}$ ромбовидно.

[2] Тождество (с немецкого)

Определение (β-редуцируемость)**.** Рефлексивное, транзитивное замыкание отношения $\to_\beta$, обозначается $\twoheadrightarrow_\beta$.

Теорема 1 (Чёрча-Россера)**.** β-редуцируемость обладает ромбовидным свойством.

Определение. $\rightrightarrows_\beta$ — параллельная β-редукция, выполняется если:

0. $A =_\alpha B$
1. $A \equiv P\ Q, B \equiv R\ S$ и $P \rightrightarrows_\beta R$ и $Q \rightrightarrows_\beta S$.
2. Аналогично β-редукции.
3. Аналогично β-редукции.

Лемма 1. $(\rightrightarrows_\beta)$ обладает ромбовидным свойством.

Лемма 2. Если R обладает ромбовидным свойством, то R^* обладает ромбовидным свойством.

Доказательство. Две индукции. □

Лемма 3. $(\rightrightarrows_\beta) \subseteq (\twoheadrightarrow_\beta)$

Доказательство теоремы Чёрча-Россера. Заметим, что:

1. $(\rightrightarrows_\beta)^* \subseteq (\twoheadrightarrow_\beta)$ — из леммы
2. $(\twoheadrightarrow_\beta) \subseteq (\rightrightarrows_\beta)^*$ — из определения
3. Т.к. $(\rightrightarrows_\beta)^*$ обладает р.с., то и $(\twoheadrightarrow_\beta)$ обладает р.с.

□

Следствие 1.1. У λ-выражения существует не более одной нормальной формы.

Доказательство. Пусть A имеет две нормальные формы: $A \twoheadrightarrow_\beta B, A \twoheadrightarrow_\beta C$ и $B \neq_\alpha C$. Тогда есть D: $B \twoheadrightarrow_\beta D$ и $C \twoheadrightarrow_\beta D$. Противоречие. □

Определение. Нормальный порядок редукции — редуцируем самый левый редекс.

Теорема 2. Если нормальная форма существует, она может быть получена нормальным порядком редукции.

Примечание. Нижеследующее объяснение — с практики.

Рассмотрим $Y\ f\ =_\beta\ f\ (Y\ f)\ =_\beta\ f\ (f\ (Y\ f))\ =_\beta\ \ldots$. Можно считать, что у f сколько угодно аргументов, первый аргумент можно считать указателем на свой рекурсивный вызов.

Пример. Числа Фибоначчи:

```
fib a b n =
    if n = 0 then a
    else fib b (a + b) (n - 1)
```

Здесь решение уравнения заметано под ковер, в λ-исчислении оно видно:

$$\text{Fib} = \lambda f.\lambda a.\lambda b.\lambda n.(\text{IsZero}\ n)\ a\ (f\ f\ a\ (a+b)\ (n-1))$$

Здесь f передается само себе, чтобы иметь ссылку на себя для рекурсивного вызова.

Для работы Fib нужно дать его самому себе: Fib Fib $1\ 1\ 10$.

Лекция 3

21 сентября

В λ-исчислении можно сделать:

1. Целые числа, где $\langle a, b\rangle \leftrightarrow a - b$
2. Рациональные числа в виде дробей
3. Матлогику?

Попытки сделать матлогику всегда приводили к парадоксам.

Оказывается, нельзя относиться к любому выражению как к логическому.

Обозначение. $\supset$ — импликация

Пример. Рассмотрим комбинатор $\Phi_A =_\beta A \supset \Phi_A$. Это $Y\ (\lambda f.\lambda a.a \supset f\ a)$.

Добавим аксиому $(A \supset (A \supset B)) \supset (A \supset B)$. Если такой аксиомы нет, то теория грустная.

Мы также хотим, чтобы если $X =_\beta Y$, то $X \supset Y$.

Каким-то образом мы получим парадокс.

3 Просто-типизированное λ-исчисление

Определение (типовые переменные).

- α, β, γ — атомарные
- τ, σ — составные

2 традиции:

1. Исчисление по Чёрчу

2. Исчисление по Карри

Мы сначала рассмотрим исчисление по Карри.

3.1 Исчисление по Карри

Типизация: $\Gamma \vdash A : \tau, \Gamma = \{x_1 : \tau_1, x_2 : \tau_2 \dots\}$

Правила:

1. $\dfrac{}{\Gamma, x_1 : \tau_1 \vdash x_1 : \tau_1}$ Ax.
2. $\dfrac{\Gamma \vdash A : \sigma \to \tau \qquad \Gamma \vdash B : \sigma}{\Gamma \vdash A\, B : \tau}$
3. $\dfrac{\Gamma, x : \tau \vdash A : \sigma}{\Gamma \vdash \lambda x.A : \tau \to \sigma}$

Пример.

$$\lambda f^{\alpha \to \alpha}.\lambda x^{\alpha}.f\ (f\ x) : (\alpha \to \alpha) \to \alpha \to \alpha$$

Подгоним доказательство под результат:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{f : \alpha \to \alpha \vdash f : \alpha \to \alpha} \qquad \dfrac{\dfrac{}{\Gamma \vdash f : \alpha \to \alpha} \qquad \dfrac{}{\Gamma \vdash x : \alpha}}{\Gamma \vdash f\ x : \alpha}}{f : \alpha \to \alpha, x : \alpha \vdash f\ (f\ x) : \alpha}}{f : \alpha \to \alpha \vdash \lambda x.f\ (f\ x) : \alpha \to \alpha}}{\lambda f.\lambda x.f\ (f\ x) : (\alpha \to \alpha) \to (\alpha \to \alpha)}$$

Теорема 3. Если $\Gamma \vdash A : \tau$, то любое подвыражение имеет тип.

Доказательство. По индукции по длине.

База. Это правило 1.

Переход. Пусть любое выражение длиной $< n$ символов обладает искомым свойством. Покажем искомое для $A : |A| = n$. Рассмотрим варианты того, по какому правилу доказана типизируемость A:

1. Второе правило: B и C короче A, следовательно для них искомое верно.
2. Третье правило: аналогично для x, B

$\square$

Теорема 4 (Subject reduction, о редукции). Если $\Gamma \vdash A : \sigma$ и $A \twoheadrightarrow_\beta B$, то $\Gamma \vdash B : \sigma$

Доказательство. Скучно.

Самая интересная часть: рассмотрим $A \to_\beta B$. Случаи:

1. $\lambda x.A \to \lambda x.B$ — индукция
2. $A\ B$ — индукция
3. $(\lambda x.A)\ B \to A[x := B]$

   По теореме о типизации подвыражений, $(\lambda x^{\tau\to\sigma}.A^\sigma)\ B^\tau\ :\ \sigma$. Кроме того, доказывается $(A[x := B]) : \sigma$.

$\square$

Лемма 4. Если $\Gamma, x : \tau \vdash A : \sigma$,$\Gamma \vdash B : \tau$, то $\Gamma \vdash A[x := B] : \sigma$

Теорема 5 (Чёрча-Россера). Если $\Gamma \vdash M : \sigma$ и существуют $N, P : M \twoheadrightarrow_\beta N, M \twoheadrightarrow_\beta P$, то найдется такой S, что $\Gamma \vdash S : \sigma$ и $N \twoheadrightarrow_\beta S$ и $P \twoheadrightarrow_\beta S$

3.2 Исчисление по Чёрчу

Язык:

- x — переменная
- $A\ B$ — аппликация
- $\lambda x^\tau.P$ — абстракция

Обозначение. Когда нужно различить исчисления, будем писать $\vdash_{\text{ч}}$ или $\vdash_{\text{к}}$

Теорема 6. Если контекст Γ и выражение P типизируется, то $\Gamma \vdash_{\text{ч}} P : \sigma$

Пример.

$$\vdash_{\text{к}} \lambda x.x : \alpha \to \alpha$$

$$\vdash_{\text{к}} \lambda x.x : \beta \to \beta$$

$$\vdash_{\text{ч}} \lambda x^\sigma.x : \sigma \to \sigma$$

Лекция 4

28 сентября

3.3 Противоречивость нетипизированного λ-исчисления

???

1. Логические выражения
2. Запрещенные выражения

Y явно нехорошее выражение. $\Phi_A =_\beta \Phi_A \supset A$

Добавим очевидные аксиомы:

1. $A =_\beta B$, то $\vdash A \supset B, \vdash B \supset A$. Почему? Потому что мы хотим, чтобы $\sin 0 = 0$, а не только $\sin 0 \to 0$
2. $(A \supset A \supset B) \supset (A \supset B)$
3. $A, A \supset B$, тогда B

Тогда заметим, что при любом $A, \vdash A$:

$$\begin{aligned}
&\Phi_A \supset \Phi_A \\
&\Phi_A \supset (\Phi_A \supset A) \\
(\Phi_A \supset (\Phi_A \supset A)) \supset {}&(\Phi_A \supset \Phi_A) \\
&\Phi_A \supset A \\
&\Phi_A \\
&A
\end{aligned}$$

4 Изоморфизм Карри-Ховарда

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}\ x \notin \Gamma$$

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}\ x \notin \Gamma$$

$$\frac{}{\Gamma, \tau \vdash \tau}$$

$$\frac{\Gamma \vdash A : \sigma \to \tau \qquad \Gamma \vdash B : \sigma}{\Gamma \vdash A\,B : \tau}$$

$$\frac{\Gamma \vdash \sigma \to \tau \qquad \Gamma \vdash \sigma}{\Gamma \vdash \tau}$$

$$\frac{\Gamma, x : \sigma \vdash A : \tau}{\Gamma \vdash \lambda x.A : \sigma \to \tau}\ x \notin \Gamma$$

$$\frac{\Gamma, \sigma \vdash \tau}{\Gamma \vdash \sigma \to \tau}$$

Теорема 7 (об изоморфизме Карри-Ховарда).

1. $\Gamma \vdash_{\lambda\to} A : \tau$, то $|\Gamma| \vdash_{\to} \tau$
2. Если $\Delta \vdash_{\to} \tau$, то найдутся $\Gamma, A : |\Gamma| = \Delta, \Gamma \vdash A : \tau$

Определение.

$$|\Gamma| \coloneqq \{\tau \mid x : \tau \in \Gamma\}$$

4.1 Импликационный фрагмент ИИВ

У нас есть только импликация.

Есть три правила: $I_{\to}, E_{\to}$, Ax. Будет ли у нас всё работать? Утверждается, что да.

Обозначение. Доказуемость в И.Ф. ИИВ будем обозначать $\Gamma \vdash_{\to} \tau$

Определение (язык И.Ф. ИИВ).

$$\tau ::= \alpha \mid (\tau \to \tau)$$

Теорема 8. Импликационный фрагмент ИИВ замкнут относительно доказуемости: Если $\Gamma \vdash \tau$ и τ содержит только пропозиционные переменные и импликацию, то $\Gamma \vdash_{\to} \tau$. Обратное очевидно верно.

Доказательство. Рассмотрим Γ^* — множество формул, замкнутых по доказуемости.

$$\tau \in \Gamma^* \Leftrightarrow \Gamma^* \vdash \tau$$

Обозначение. Γ — множество формул, тогда Γ^* — замыкание этого множества по доказуемости, а $\Gamma^{\overset{*}{\to}}$ — замыкание по доказуемости в ИФИИВ.

Рассмотрим множество миров: $\Gamma^{\overset{*}{\to}} \preceq \Delta^{\overset{*}{\to}}$, если $\Gamma^{\overset{*}{\to}} \subseteq \Delta^{\overset{*}{\to}}, \Delta^{\overset{*}{\to}}$ — замкн., $\Gamma^* \Vdash \tau$, если $\tau \in \Gamma^*$

Утверждение. Γ^* образует модель Крипке.

Определение (модель Крипке).

1. Множество миров, упорядоченных отношением $\preceq$
2. $\Vdash$ такое, что если $\Gamma \Vdash \alpha$, то $\Gamma \preceq \Delta$, то $\Delta \Vdash \alpha$.

Тогда $\Gamma \Vdash \tau \to \sigma$ тогда и только тогда, когда в любом $\Gamma \preceq \Delta$ из $\Delta \Vdash \tau$, следует $\Delta \Vdash \sigma$.

Утверждение. $\tau \in \Gamma^{\overset{*}{\to}}$ тогда и только тогда, когда $\Gamma^{\overset{*}{\to}} \vdash_{\text{и}} \tau$

Доказательство. Индукция по структуре τ.

База. $\tau \equiv \alpha$

$\Rightarrow$ $\alpha \in \Gamma^{\overset{*}{\to}}$, то $\alpha \vdash_{\text{и}} \alpha$

$\Leftarrow$ $\alpha \vdash_{\text{и}} \alpha$, тогда очевидно $\alpha \in \Gamma^{\overset{*}{\to}}$

Переход. $\tau \equiv \delta \to \pi$

$\Rightarrow$ $\sigma \to \pi \in \Gamma^{\overset{*}{\to}}$, то $\Gamma^{\overset{*}{\to}} \vdash_{\to} \sigma \to \pi$

$\Leftarrow$ $\Gamma^{\overset{*}{\to}} \vdash_{\text{и}} (\sigma \to \pi)$. Значит, $\Gamma^{\overset{*}{\to}} \Vdash \sigma \to \pi$.

$\square$

Рассмотрим $\Gamma^{\overset{*}{\to}} \preceq \Delta : \Delta \Vdash \sigma$, то $\Delta \Vdash \pi$. Значит, $\Delta \vdash \sigma$. Значит, $\sigma \in \Delta$, т.е. $\Delta \vdash_{\to} \sigma$. Значит, $\Delta \vdash_{\to} \pi$ по М.Р., т.к. $\Gamma^{\overset{*}{\to}} \Vdash \sigma \to \pi \Rightarrow \Delta \Vdash \sigma \to \pi \Rightarrow \Delta \vdash_{\to} \sigma \to \pi$

Утверждение. $\Gamma \Vdash \tau \Leftrightarrow \Gamma|_{\to}\tau$

Доказательство.

$\Rightarrow$ $\Gamma \Vdash \tau$.

1. $\Gamma \Vdash \alpha$, т.е. $\alpha \in \Gamma$, т.е. $\alpha \vdash_{\to} \alpha$
2. $\Gamma \Vdash \sigma \to \pi$.

   Рассмотрим $\Gamma \preceq \Delta$, причём $\Delta \Vdash \sigma$, тогда $\Delta \Vdash \pi$. Т.е. по индукционному предположению $\Delta \vdash_{\to} \sigma$. Пусть $\Delta = \{\Gamma, \sigma\}^*$. Тогда $\Gamma, \sigma \vdash_{\to} \sigma$.

Тогда $\Gamma, \sigma \Vdash \pi$ по индукционному предложил и определению $\Vdash$. Тогда $\Gamma, \sigma \vdash_{\to} \pi$, т.е. $\Gamma \vdash_{\to} \sigma \to \pi$

$\Leftarrow$ $\Gamma \vdash_{\to} \tau$, тогда $\Gamma \Vdash \tau$

1. $\tau \equiv \alpha$ — очевидно.
2. $\tau \equiv \sigma \to \pi$. Дано, что $\Gamma_{\to} \vdash \sigma \to \pi$.

   Пусть $\Delta \Vdash \sigma$. $\Gamma \preceq \Delta$. Тогда $\Delta \vdash_{\to} \sigma$ по индукционному предположению. $\Gamma \vdash_{\to} \sigma \to \pi$, т.е. $\Delta \vdash_{\to} \sigma \to \pi$. По М.Р. $\Delta \vdash_{\to} \pi$. По индукционному предположению $\Delta \Vdash \pi$. Т.е. $\Gamma \Vdash \sigma \to \pi$. В лекции было $\models$.

$\square$

Схема доказательства:

1. $\tau \in \Gamma^*$, если $\Gamma^* \vdash_{\text{и}} \tau$
2. $\Gamma^* \Vdash \tau$
3. $\Gamma^* \Vdash \tau$ тогда и только тогда, когда $\Gamma^* \vdash_{\to} \tau$

$\square$

Обозначение. $\lambda_{\to}$ — типизированное λ-исчисление.

1. Обитаемость: $\overset{?}{\Gamma} \vdash ? : \tau$ — по изоморфизму Карри-Ховарда и теореме об эквивалентности $\Gamma \vdash \tau$
2. Вывод (реконструкция): $\Gamma \vdash A :?$
3. Проверка: $\Gamma \vdash A : \tau$

Пункты 2 и 3 это одно и то же.

Лекция 5

5 октября

5 Алгебраические термы

Определение (алгебраические термы).

$$T ::= \underbrace{a}_{\text{переменная}} \quad | \quad \underbrace{f}_{\substack{\text{функциональный}\\ \text{символ}}} \quad T_1 \dots T_n$$

Функциональные символы $\in F$, переменные $\in T$

Пример. $f\ (f_2\ a\ b)\ c$

Определение. **Подстановка переменных** — отображение $S_0 : V \to T$, являющееся тождественным почти всюду[1], то есть $\exists$ фиксированные $a_1 \dots a_n$, для которых S_0 не тождественна: $S_0(a_i) = T_i$, а для $b \notin \{a_i\}$ $S_0(b) = b$.

Тогда можно определить определить подстановку $S : T \to T$:

$$S(f\ T_1 \dots T_n) = f\ (S(T_1))\ (S(T_2)) \dots (S(T_n))$$

$$S(a) = S_0(a)$$

Определение. Рассмотрим уравнение $T_1 = T_2$. Его **решение** — такая подстановка S, что $S(T_1) \equiv S(T_2)$, где $\equiv$ обозначается равенство строк.

Пример.

$$f\ a\ (g\ b) = f\ (g\ c)\ d$$

$$S_0(a) = g\ c \quad S_0(d) = g\ b$$

$$S(f\ a\ (g\ b)) = f\ (g\ c)\ (g\ b)$$

[1] Кроме конечного количества.

Определение (система уравнений).

$$\begin{cases} T_1 = P_1 \\ T_2 = P_2 \\ \vdots \\ T_n = P_n \end{cases}$$

5.1 Эквивалентность уравнений и систем

Определение. Две системы $E_1 : \begin{cases} T_1 = P_1 \\ \vdots \\ T_n = P_n \end{cases}$ и $E_2 : \begin{cases} T_1' = P_1' \\ \vdots \\ T_n' = P_n' \end{cases}$ называются **эквивалентными**, если любое решение системы E_1 подходит к E_2 и наоборот.

Утверждение. Для любой системы существует эквивалентное уравнение.

Доказательство. Выберем новый n-местный функциональный символ h, построим уравнение $h\ T_1 \dots T_n = h\ P_1 \dots P_n$. $\square$

Определение.

$$(U \circ T)(P) = U(T(P))$$

Определение. Определим порядок на подстановках. $S \preceq T$, если S — частный случай T, т.е. $\exists U\colon S = U \circ T$

Определение. **Наиболее общим решением** уравнения $T = P$ назовём подстановку S, такую что $S(T) = S(P)$ и для любой S_1: $S_1(T) \equiv S_1(P)$ выполнено $S_1 \preceq S$

Теорема 9. У уравнения в алгебраических термах $T = P$ всегда есть наиболее общее решение, если есть хоть какое-то.

Определение. **Несовместная система** — система с уравнениями вида $f\ T_1 \dots T_n = g\ P_1 \dots P_k$, где $f \not\equiv g$, либо $x = \dots x \dots$.

В `OCaml` и `Haskell` это называется “occurs check”.

Определение. Система в **разрешённой форме** — система вида $\begin{cases} a_1 = T_1 \\ \vdots \\ a_n = T_n \end{cases}$, где:

1. Все a_i различны
2. T_i не содержит a_j для $i \neq j$

Альтернативное определение — каждый a_i входит по одному разу.

5.2 Алгоритм унификации

Рассмотрим систему $\begin{cases} T_1 = P_1 \\ \vdots \\ T_n = P_n \end{cases}$

Применяем одно из следующих:

1. $x = x$ — отбрасываем.
2. $T = x$, где $T \not\equiv x$, тогда заменяем на $x = T$
3. $\begin{cases} x = P \\ \vdots \\ T_2 = P_2 \\ \vdots \\ T_n = P_n \end{cases} \Rightarrow \begin{cases} T_2[x := P] = P_2[x := P] \\ \vdots \\ T_n[x := P] = P_n[x := P] \\ x = P \end{cases}$
4. $f\ T_1 \ldots T_n = f\ P_1 \ldots P_n \Rightarrow \begin{cases} T_1 = P_1 \\ \vdots \\ T_n = P_n \end{cases}$

Теорема 10. Применяя шаги алгоритма унификации, за конечное время можно получить систему либо в разрешенной форме, либо несовместную.

Доказательство. Рассмотрим тройку $\left\langle \substack{\text{количество}\\\text{неразрешенных}\\\text{переменных}}, \substack{\text{максимальная}\\\text{сложность}\\\text{слева}}, \substack{\text{количество}\\\text{уравнений}\\\text{максимальной}\\\text{сложности}\\\text{слева}} \right\rangle$. Сложность — вложенность.

1. Применения правил уменьшают тройку.
2. $\langle 0, 0, t \rangle$ — система в разрешенной форме.
3. Количество применений правил конечно, т.к. каждая из троек $\in \omega^3$ и любая последовательность убывающих ординалов конечна.

$\square$

5.3 Вывод типов в $\lambda_\rightarrow$

$(\rightarrow)$ — 2-местный функциональный символ.

Проведём индукцию по структуре λ-выражения. Результатом разбора будет пара $\langle$система, тип$\rangle$

5.3.1 Построение уравнений

Структура	Комментарий	Система	Тип
x	Введём тип α_x	$\varnothing$	α_x
$A\ B$	Рекурсивный вызов на A и B даст $\langle E_A, \tau_A\rangle, \langle E_B, \tau_B\rangle$. Вводим β — новый тип.	$E_A, E_B, \tau_B \to \beta = \tau_A$	β
$\lambda x.A$	Рекурсивный вызов на A даст $\langle E_A, \tau_A\rangle$. Берём тип для $x : \alpha_x$.	E_A	$\alpha_x \to \tau_A$

Несложно заметить, что эти правила соответствуют правилом вывода в типизированном λ-исчислении.

5.3.2 Разрешение системы

Это унификация.

Пример. Разберём $B = \lambda x.\overbrace{\ x\ }^{A}$.

- $E_A = \varnothing, \tau_A = \alpha_x$
- $E_B = \varnothing, \tau_B = \alpha_x \to \alpha_x$

Разрешим систему уравнений τ_A, τ_B. Оказывается, эта система уже в разрешенной форме. Таким образом, $\vdash \lambda x.x : \alpha \to \alpha$. Контекст пустой, т.к. свободных переменных нет ($E_A, E_B = \varnothing$).

Определение. Терм называется **слабо-нормализуемым**, если существует последовательность β-редукций, приводящих его в нормальную форму.

Определение. Терм называется **сильно-нормализуемым**, если не существует бесконечной последовательности β-редукций, нигде не приводящая его в нормальную форму.

Пример. $K\ I\ \Omega$ — слабо нормализуемый, но не сильно нормализуемый

Теорема 11. $\lambda_\to$ сильно нормализуемо.

Примечание. Это сильно ограничивает выразительность $\lambda_\to$.

6 Исчисление предикатов второго порядка

Второй порядок — это когда переменные есть предикаты.

Определение (предикат).

$$\Phi_\Pi ::= p \mid \Phi_\Pi \cup \Phi_\Pi \mid \Phi_\Pi \,\&\, \Phi_\Pi \mid \Phi_\Pi \to \Phi_\Pi \mid \forall p.\Phi_\Pi \mid \exists p.\Phi_\Pi$$

Утверждение. Можно выразить:

$$\begin{aligned} a \mathbin{\&} b &::= \forall p.(a \to b \to p) \to p \\ a \vee b &::= \forall p.(a \to p) \to (b \to p) \to p \\ \perp &::= \forall p.p \\ \exists p.A &::= \forall x.(\forall p.p \to x) \to x \end{aligned}$$

Это исчисление также называется “Система F”, оно же L_2.

$$L_2 ::= x \mid \lambda x^{\alpha}.A \mid P\ Q \mid P\tau \mid \lambda\alpha.A$$

Лекция 6

12 октября

7 Абстрактные типы данных

ООП = АТД + наследование.

Пример (стек).

$$\texttt{push} : \alpha \to \alpha\ \texttt{stack} \to \alpha\ \texttt{stack}$$

$$\texttt{pop} : \alpha\ \texttt{stack} \to (\alpha \cdot \alpha\ \texttt{stack})$$

$$\texttt{empty} : \alpha\ \texttt{stack}$$

Что мы понимаем под $\exists\alpha.\varphi$? φ — интерфейс и существует где-то в природе тип, который этому интерфейсу соответствует.

Для стека:

$$\exists\alpha.\ \underbrace{(\eta \to \alpha \to \alpha)}_{\texttt{push}} \,\&\, \underbrace{(\alpha \to \alpha \,\&\, \eta)}_{\texttt{pop}} \,\&\, \underbrace{\eta}_{\texttt{empty}}$$

Правила вывода:

$$\frac{\Gamma \vdash \varphi[x := \theta]}{\exists x.\varphi}$$

$$\frac{\Gamma, \psi \vdash \varphi \qquad \Gamma \vdash \exists x.\psi}{\Gamma \vdash \varphi}\ x \notin \mathrm{FV}(\Gamma)$$

$$\frac{\Gamma \vdash M : \sigma[\alpha := \tau]}{\Gamma \vdash \texttt{pack}\ \tau, M\ \texttt{to}\ \exists\alpha.\sigma : \exists\alpha.\sigma}$$

TBD.

Лекция 7

19 октября

8 Типовая система Хиндли-Милнера

Мы рассмотрели две системы типов:

1. Просто типизированное лямбда исчисление: недостаточно выразительно
2. Система F: местами выразительна, местами недостаточно. Кроме того, потеряна разрешимость.

Ограничим излишнюю свободу системы F.

Определение (ранг типа). Пусть σ — тип без кванторов. $R \subset \text{тип} \times \mathbb{N}_0$, такое что:

1. $R(\sigma, 0)$
2. Если $R(\tau, k)$, то $R(\forall\alpha.\tau, \max(k, 1))$
3. Если $R(\tau_0, k)$ и $R(\tau_1, k+1)$, то $R(\tau_0 \to \tau_1, k+1)$.

Пример.

$$R(\alpha, 0) \Rightarrow R(\alpha, 5) \Rightarrow R(\forall\alpha.\alpha, 5)$$
$$R(\alpha \to \alpha, 0) \Rightarrow R(\forall\alpha.\alpha \to \alpha, 1)$$

Определение (Типовая система Хиндли-Милнера). Рассмотрим λ-исчисление 2 порядка по Карри.

Типы:

1. Типы без кванторов: $\tau = \alpha \mid (\tau \to \tau)$
2. Типовые схемы: $\sigma = \forall\alpha.\sigma \mid \tau$

Определение (Отношение “**быть частным случаем**” (специализация)). $\sigma_1 \sqsubseteq \sigma_2$ (σ_2 — частный случай σ_1), если $\sigma_1 \equiv \forall \alpha_1 \dots \forall \alpha_n . \tau_1$, $\sigma \equiv \forall \beta_1 \dots \beta_k . \tau_1[\alpha_1 \coloneqq \theta_1 \dots \alpha_n \coloneqq \theta_n]$ и новые $\beta_1 \dots \beta_k$ не входят свободно в σ_1.

Пример. $\tau \sqsubseteq \mathtt{string}$

$$\begin{tikzcd}
 & \mathtt{string} & \\
\gamma \arrow[ur, bend left, "{\gamma[\gamma:=\mathtt{string}]}"] \arrow[r, "{\gamma[\gamma:=\alpha\to\beta]}"] \arrow[dr, bend right] & \alpha \to \beta \arrow[r] & \mathtt{string} \to \mathtt{int} \\
 & (\alpha \to \beta) \to (\mathtt{string} \to \mathtt{int}) &
\end{tikzcd}$$

Пример.

$$\forall \alpha . \alpha \to \alpha \sqsubseteq \forall \beta . (\beta \to \beta) \to (\beta \to \beta)$$

Правила вывода:

$$\frac{}{\Gamma, x : \sigma \vdash x : \sigma}\ \alpha \notin \mathrm{FV}(\Gamma)$$

$$\frac{\Gamma \vdash A : \tau \to \tau' \qquad \Gamma \vdash B : \tau}{\Gamma \vdash A\,B : \tau'}$$

$$\frac{\Gamma, x : \tau \vdash A : \tau'}{\Gamma : \lambda x . A : \tau \to \tau'}$$

$$\frac{\Gamma \vdash A : \sigma \qquad \Gamma, x : \sigma \vdash B : \tau}{\Gamma \vdash \text{let } x = A \text{ in } B : \tau}$$

$$\frac{\Gamma \vdash A : \sigma'}{\Gamma \vdash A : \sigma}\ \sigma' \sqsubseteq \sigma$$

$$\frac{\Gamma \vdash A : \sigma}{\Gamma \vdash A : \forall \alpha . \sigma}\ \alpha \notin \mathrm{FV}(\Gamma)$$

$$\text{let } x = A \text{ in } B \twoheadrightarrow_\beta B[x \coloneqq A]$$

Казалось бы, let похож на $(\lambda x . B)\ A$. Однако, мы разрешаем кванторы в A.

Пример.

$$I \equiv \lambda x . x : \forall \alpha . \alpha \to \alpha$$

$$\lhd(I\ 1, I\ \mathtt{"a"}) \quad I\ 1 : \mathtt{int}, I\ \mathtt{"a"} : \mathtt{string}$$

1. $$\text{let } \underbrace{I = \lambda x . x}_{I : \forall \alpha . \alpha \to \alpha} \text{ in } (I\ 1, I\ \mathtt{"a"})$$

2. То же самое, но без let:
$$(\lambda i.(i\ 1, i\ \texttt{"a"}))\ (\lambda x.x)$$
В этом варианте тип внутри лямбды без кванторов. В силу этого операцию $(i\ 1, i\ \texttt{"a"})$ сложно выполнить — нужно угадать, какой тип подставлять.

Эта система очевидно у́же, чем System F. Мы её сузили, чтобы получить разрешимость.

8.1 Алгоритм W

Мы хотим решить $? \vdash A\ :?$, т.е найти контекст и тип выражения, притом наиболее общие.

$W(\Gamma, E) \Rightarrow (\tau, S)$ — по контексту и выражению получаем тип и подстановку.

1. $E \equiv x, x \in \Gamma, x : \sigma_x = \forall \alpha_1 \dots \alpha_n.\tau$. Тогда введём новые переменные $\beta_1 \dots \beta_n$ и результатом будет:
$$W(\Gamma, E) = (\forall \beta_1 \dots \beta_n.\tau, \varnothing)$$
2. $E \equiv \lambda x.P$. Пусть $W(\Gamma \cup \{x : \gamma\}, P) = (\tau_P, S_1)$.
$$W(\Gamma, E) = (S_1(\gamma \to \tau_P), S_1)$$
3. $E \equiv P\ Q$. Введём новый тип γ. Пусть $\mho$ — вызов алгоритма унификации и:
$$W(\Gamma, P) = (\tau_P, S_1) \quad W(S_1\Gamma, Q) = (\tau_Q, S_2) \quad \mho(S_2\tau_P, \tau_Q \to \gamma) = S_3$$
Тогда:
$$W(\Gamma, E) = (S_3\gamma, S_3 \circ S_2 \circ S_1)$$
4. $E \equiv \text{let } x = P \text{ in } Q$. Пусть:
$$W(\Gamma, P) = (\tau_P, S_1) \quad W(S_1\Gamma \cup \{x : \forall^{1}.\tau_f\}, Q) = (S_2, \tau_Q)$$
$$W(\Gamma, E) = (\tau_Q, S_2 \circ S_1)$$

Пример.
$$\text{let } I = \lambda x.x \text{ in } (I\ 1, I\ \text{``''})$$
Согласно 4 пункту алгоритма:
$$I : \forall \alpha.\alpha \to \alpha \vdash (I\ 1, I\ \text{``''})$$

Мы теряем полноту по Тьюрингу, т.к. это частный случай системы F. Мы такого не хотим, поэтому добавим чего-нибудь, что её нам даст.

[1] Кванторы по всем свободным в τ_f переменным.

1. Правило для Y:

$$Y : \forall\alpha.(\alpha \to \alpha) \to \alpha$$

   Теория становится противоречивой — вместо $\alpha \to \alpha$ всегда можно подставить id и получить любое α.

2. $\triangleleft$`IntList` $=$ (`int` & `IntList`) $\vee$ `Nil`. Это какое-то уравнение. Как его решить?

   Введём μ-оператор: $\mu\,\eta.(\texttt{int}\,\&\,\eta) \vee \texttt{Nil}$ или в общем случае $\mu\,\eta.\tau$ — тип, решающий уравнение $\eta = \tau$

Есть две традиции решения таких уравнений:

1. Экви-рекурсивные: μ существует как тип (Java).
2. Изо-рекурсивные: ищем $\mu\eta.\tau(\eta)$ вводятся две операции
   (a) Roll: $\tau(\eta) \to \eta$
   (b) Unroll: $\eta \to \tau(\eta)$

Итого мы взяли систему F и:

1. Запретили типы с неповерхностными кванторами
2. Добавили `let`-полиморфизм
3. Добавили противоречивость через Y-комбинатор и решение уравнений на типах.

Лекция 8

26 октября

Это последняя лекция, посвященная части “Матлогика в языках программирования”. Вторая часть — “Языки программирования в матлогике и математике”.

9 λ-куб

Мы упустили теорию первого порядка.

9.1 Обобщенные типовые системы

$$\mathcal{F} ::= x \mid \underbrace{\mathcal{F}\,\mathcal{F}}_{\text{применение}} \mid \underbrace{\lambda x : \mathcal{F}.\mathcal{F}}_{\lambda-\text{абстракция}} \mid \Pi x : \mathcal{F}.\mathcal{F} \mid \underbrace{C}_{\text{константа}}$$

$$C ::= * \mid \square$$

Мы решили, что типы и выражения должны сосуществовать, например в C++ можно написать `Array<int, 23+7>`.

Обозначение. $s \coloneqq$ множество $(*, \square)$

$$\frac{}{\vdash * : \square}\ \text{аксиома}$$

$$\frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}\ \text{начальное правило, } x \notin FV(\Gamma)$$

$$\frac{\Gamma \vdash \varphi : (\Pi x^A.B) : s \qquad \Gamma \vdash a : A}{\Gamma \vdash (\varphi\ a) : (B[x \coloneqq A])}\ \text{применение}$$

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s \qquad B =_\beta B'}{\Gamma \vdash A : B'}\ \text{преобразование}$$

$$\frac{\Gamma \vdash B : C \qquad \Gamma \vdash A : s}{\Gamma, x : A \vdash B : C}\ \text{ослабление, } x \notin FV(\Gamma)$$

$*$ — тип, $\square$ — тип типа.

Пример. `array [a..b] of T`. Можно рассматривать `array [a..b] of` как оператор над типами. Его тип $* \to *$. Это также называется не тип, а род.

Примечание.

$$\underset{\substack{\text{знач.}\\ value}}{7} : \underset{\substack{\text{тип}\\ type}}{int} : \underset{\substack{\text{род}\\ kind}}{*} : \underset{\substack{\text{сорт}\\ sort}}{\square}$$

Пусть $S \subseteq C \times C$ параметризует типовую систему. Здесь и далее (s_1, s_2) пробегает все пары $\in S$.

$$\frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\Pi x^A.B) : s_2}\ \Pi\text{-правило}$$

$$\frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash b : B \qquad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\lambda x^A.b) : (\Pi x^A.B)}\ \lambda\text{-правило}$$

Обозначение. Будем писать $\Pi x^{\varphi}.\pi$ вместо $\varphi \to \pi$, если $x \notin FV(\pi)$.

Обозначение.

$$x : y : z \Rightarrow x : y, y : z$$

Примечание. Пусть $x \notin FV(B)$. Тогда рассмотрим следующее правило:

$$\frac{\Gamma \vdash A : * \qquad \Gamma, x : A \vdash b : B \qquad \Gamma, x : A^{[1]} \vdash B : *}{\Gamma \vdash (\lambda x^A.b) : A \to B}$$

А что если $x \in FV(B)$? Тогда мы получаем **зависимый тип**.

Пример. `sprintf("%d", "a")` — так нельзя.

`sprintf` : $(x :$ `string`$) \to F(x)$ — так мы не пишем, не сложилось по традиции. Мы будем писать $\Pi x^{\texttt{string}}.F(x)$

`sprintf "%s"` : `string` $\to$ `string`

Рассмотрим S из определения.

[1] Можно убрать, т.к. $x \notin FV(B)$

S	Название системы типов	Характерный представитель
$(*, *)$	$\lambda_{\to}$	$\lambda x.x$
$(*, *), (\square, *)$	$\lambda_{\to}$	$\Lambda\alpha.\lambda x^{\alpha}.x$
$(*, *), (\square, *)$	$\lambda\,\omega$ слабая	`Int[]`
$(*, *), (\square, *), (\square, \square)$	$\lambda\,\omega$	
$(*, *), (*, \square)$		`int[n]`
$(*, *), (*, \square), (\square, *), (\square, \square)$	λC : исчисление конструкций[2]	

Объектно-ориентированное программирование не описывается через S.

Дальше в лекции были примеры, которые не записаны.

[2] Языки доказательств

Лекция 9

2 ноября

10 Гомотопическая теория типов

Первые полчаса лекции пропущены.

Определение. Путь между a и b в пространстве X — функция $f : [0,1] \to X, f(0) = a, f(1) = b$ и f непрерывно. Если есть путь из a в b, то мы считаем a и b равными.

Определение. Интервальный тип: $I = [left, right]$

Почему не $I = \{left, right\}$?

Пример. Докажем, что $2 + 1 = 1 + 2$. Рассмотрим f — потенциальный путь $f(left) = 1 + 2, f(right) = 2 + 1$

Определение. Отображение **непрерывно**, если[1] прообразо открытого множества открыт.

Пример. $f : \{0,1\} \to \{2,3\}$, дискретная топология. Непрерывна ли $fx = x + 2$?

Открытые в $\{2,3\} : \{\}, \{2\}, \{3\}, \{2,3\}$

$$f^{-1}(\{\}) = \{\}, f^{-1}(\{2\}) = \{0\}, f^{-1}(\{3\}) = \{1\}, f^{-1}(\{2,3\}) = \{0,1\}$$

Таким образом f непрерывна.

Пример. $\lhd f : \mathbb{R} \to \mathbb{R}, f(x) = \lfloor x \rfloor$

$f^{-1}((0.1, 0.2)) = 0$, что не является открытым множеством.

Кроме того, $f^{-1}([0,5]) = \{0,1,2,3,4,5\}$

[1] И только если.

Определение. Пространство X **не связно**, если существует $P, Q : X = P \cup Q, P \cap Q = \varnothing, P \neq X, Q \neq X, P, Q$ открыты.

Множество **связно**, если является связным пространством под индуцированной топологией (или при сужении топологии на него, то же самое).

Пример. $(0,1) \cup (2,3)$ — не связно.

Пример. $(0,1) \cup (2,3]$ — тоже не связно, т.к. $(2,3]$ открыто в $(0,1) \cup (2,3]$

Пример. $(0,1) \cup (2,3)$ в дискретной топологии — не связно.

Упражнение 1. Предложите топологию, в которой пространство $(0,1)+(2,3)$ связно. Тривиальная топология.

Определение. Пространство **линейно связно**, если существует путь из любой точки в любую.

Первое определение гласит, что мы не можем провести границу, а второе — что не можем ???.

Определение. **Равенство** — тип пути из $a : X$ в $b : X$. Равенство обитаемо, если такой путь существует.

Такое определение сложно уместить в язык программирования, т.к. $[0,1]$ не влезает в компьютер, поэтому придуман интервальный тип $I = [left, right]$.

Пример. $\lhd 1 : \text{Nat}, 2 : \text{Nat}$, в Nat дискретная топология. Нужно построить $f : f(0) = 1, f(1) = 2$.

Рассмотрим `Path` в Arend:

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

I — интервальный тип, A — отображение интервал $\to$ тип. path — единственный конструктор, принимает функцию, сопоставляющую точки интервала значение A в этой точке.

Докажем в Arend, что $1 = 1$:

```
\func oneone : (1 = 1) => idp
```

Если мы не знаем, что сделать, мы можешь написать ?, то выражение условно принимается. Иногда работают рефакторинги, которые заменят ? на доказательство.

Фигурные скобки вокруг аргумента обозначают, что аргумент неявный. Запишем в явном виде:

```
\func oneone : (1 = 1) => idp {Nat} {1}
```

`oneone` — значение зависимого типа равенства.

```
\func arar : ((1 Nat.+ 2) = (2 Nat.+ 1)) => idp
```

`idp` это `refl` из Lean.

У нас интенсиональное равенство.

Построим фукнцию с типовым параметром:

```
\func second (t : \Type) (ttt : \Pi (x : t) -> t) (y : t) : t => ttt y
```

Здесь `ttt` — функция $t \to t$, мы её применяем к $y : t$ и получаем $ttt\ y : t$

```
\func third (t : \Type) (ttt : \Pi (x : t) -> t) (y : t) => ttt y = ttt y
```

Лекция 10

9 ноября

11 Равенство

Напоминание: $a = b$ по определению выполнено тогда и только тогда когда существует путь $a \rightsquigarrow b$. Путь можно определить так:

Определение. Пусть I — интервальный тип. Если существует непрерывная функция $f : I \to A$, такая что f left $=_\beta a$, f right $=_\beta b$, то $a \rightsquigarrow b$.

То же самое в рамках Arend:

```
\data Path
    path (f : I → A) : f left = f right
```

Когда мы говорим о любом типе данных, у нас есть две конструкции:

1. Построение
2. Удаление

Для `Path` есть построение — конструктор. Мы хотим получить ещё и элиминатор.

Пример (элиминатор для $\vee$).

```
case (f : L → θ) (g : R → θ) (v : L ∨ R) : θ
```

Пример (элиминатор для I).

```
\func coe (P : I → \Type)
        (a : P left)
        (i : I) : P i \elim i
    | left ⇒ a
```

Т.к. все значения на пути равны, то мы возвращаем значение на `left`.

`\elim` это паттерн матчинг. Т.к. I — особый тип, нам не нужно расписывать все случаи.

Это все весьма странно, но это единственная конструкция подобного рода.

```
\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
```

Обозначение. Если `p : a = a'`, то:

1. `p @ left` это `a`
2. `p @ right` это `a'`

Примечание. Фигурные скобки означает типы, которые компилятор сам выведет из контекста. Рассчитывать на него нельзя, т.к. выведение типов неразрешимо в общем случае.

Пример (доказательство равенства).

```
\func inv (A : \Type) {a a' A} (p : a = a') : a' = a
    ⇒ transport (\lam x ⇒ x) p idp

\func idp {A : \Type} {a : A} : a = a
    ⇒ path (\lam _ ⇒ a)
```

Пример (конгруэнтность).

```
\func pmap (A B : \Type) (f : A → B) (a a' : A) {p : a = a'} : f a = f a'
    ⇒ transport (\lam x ⇒ f a = f x) p idp
```

Пример (натуральные числа).

```
\data Nat
    | zero
    | suc (k : Nat)

\data Empty

Not (A : \Type) : A → Empty

Not (zero = suc zero)

\func proof_ne (a : Nat) : \Type \elim a
    | zero ⇒ 0 = 0
    | suc x ⇒ Empty

\func zne (x : 0 = 1) : Empty
    ⇒ transport proof_ne {0} {1} x idp
```

Примечание. В стандартной библиотеке это доказано немного по-другому, вместо `0 = 0` используется `\Sigma` тип кортежа из нуля элементов, то есть `()` из Haskell.

Лекция 11

15 ноября

Все, что мы доказывали, как-то не очень интересно — это тождества. Что если мы хотим доказать например $a \leq b$? Для этого для начала надо уметь это записать.

Определение (меньше или равно). $a \leq b$ это $\exists x . a + b = b$

Несложно догадаться, что у нас экзистенциальный тип. В Arend все экзистенциальные типы это пары вида $(x : A, a + x = b : A \to \backslash\mathrm{Type})$. Таким образом, наш тип это $\backslash\mathrm{Sigma}\ (x : \mathrm{Nat})\ (a + x = b)$.

Пример. Хотим доказать $5 \leq 12$, тогда доказательство это $(7, \mathrm{idp}) : \backslash\mathrm{Sigma}\ (x : \mathrm{Nat})\ (5 + x = 12)$

Определим наш тип "$\leq$" индуктивно:

```
\data loe (a b : Nat) \with
    | zero, _ ⇒ base
    | suc a', suc b' ⇒ next (p : loe a' b')
```

Примечание. `base` и `next` — конструкторы типа, а не магическая вещь для индукции.

Пример.

- `base: loe 0 15`
- `next (next (base)): loe 1 16`

Эти два определения эквивалентны. Докажем это.

Из индуктивного в экзистенциальный тип:

```
\func f1 {a b : Nat} (p1 : loe a b) : loe' a b
    | {0}, {b}, base ⇒ (b, idp)
    | {suc a}, {suc b}, next (pr1) ⇒
        \let (pb, ppr) ⇒ f1 pr1 \in (pb, pmap suc ppr)
```

В обратную сторону:

```
\func f2 {a b : Nat} (p1 : loe' a b) : loe a b
    | {0}, _ => base
    | {suc a}, {0}, (x, p) => absurd (transport (\lam t => \case t \with {
        | 0 => Empty
        | suc n => \Sigma}) p ())
```

Это не удобно писать, поэтому напишем `contradiction`[1]. Эта конструкция умеет доказывать противоречия за 1–2 шага. Таким образом:

```
\func f2 {a b : Nat} (p1 : loe' a b) : loe a b
    | {0}, _ => base
    | {suc a}, {0}, (x, p) => contradiction
    | {suc a}, {suc b}, (x, p) => next (f2 (x, pmap minus1 p))
```

Докажем часть домашнего задания:

```
\func plus-assoc {a b c : Nat} : (a + b) + c = a + (b + c) \elim c
    | 0 => idp
    | suc c => {?}
```

Можно вместо `{?}` написать `pmap suc (plus-assoc)`, но это скучно. Можем написать `rewrite plus-assoc idp`, который докажет нам `suc ((a + b) + c) = suc (a + (b + c))`, переписав `(a + b) + c` на `a + (b + c)`, т.к. есть доказательство `(a + b) + c = a + (b + c)`. Это звучит как магия, но на самом деле делается так:

```
transport (\lam x => (a + b) + suc x = suc x) plus-assoc idp
```

12 Классы

Пример. Группа: $\langle R, +, e, ^{-1}\rangle$, такие что:

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

Попробуем описать что-то подобное в Arend.

```
\instance OrdNat : Preorder Nat
| ⩽ (a b : Nat) => TruncP (\Sigma (r : Nat) (r + a = b))
| ⩽-reflexive => inP (0, idp)
| ⩽-transitive {x} {y} {z} =>
```

[1] Для этого надо добавить `dependencies: [arend-lib]` в `arend.yaml`

```
\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 (d2 + d1, plus-assoc *> (rewrite p1 p2))
  }
```

У `TruncP` есть математическое объяснение, и есть программистское. У нас есть различные доказательства и мы их все объявляем равными.

План дальнейших лекций:

1. `Set`/`Prop`
2. Теорема Диаконеску: теория множеств + ИИВ + аксиома выбора это классическая логика.

Дальше есть несколько вариантов:

1. Гомотопическая теория типов (математически)
2. Другие языки: возможно Idris, Coq
3. Другие исчисления, возможно F_Ω, линейные типы.