Введение в Теорию Типов
Конспект лекций

Штукенберг Д. Г.
Университет ИТМО

22 января 2022 г.

Содержание

1 Лекция 1

1.1 λ-исчисление

Определение 1.1 (λ-выражение)**.** λ-выражение — выражение, удовлетворяющее грамматике:

$$\Phi ::= x \,|\, (\Phi) \,|\, \lambda x.\Phi \,|\, \Phi\ \Phi$$

Иногда для упрощения записи мы будем опускать скобки. В этом случае перед разбором выражения следует расставить все опущенные скобки. При их расставлении будем придерживаться правил:

1. В аппликации расставляем скобки слева направо: $A\ B\ C \implies (A\ B)\ C$.
2. Абстракции жадные — поглощают скобками все, что могут, до конца строки: $\lambda a.\lambda b.a\ b \implies \lambda a.(\lambda b.(a\ b))$.

   Пример. $\lambda x.(\lambda f.((fx)(fx)\lambda y.(yf)))$

Договоримся, что:

- Переменные — x, a, b, c.
- Термы (части λ-выражения) — X, A, B, C.
- Фиксированные переменные обозначаются буквами из начала алфавита, метапеременные — из конца.

Есть понятия связанного и свободного вхождения переменной (аналогично исчислению предикатов).

Определение 1.2. Если вхождение x находится в области действия абстракции по x, то такое вхождение называется связанным, иначе вхождение называется свободным.

Определение 1.3. Терм Q называется свободным для подстановки в Φ вместо x, если после подстановки ни одно свободное вхождение переменной в Q не станет связанным.

Пример. $\lambda x.A$ связывает все свободные вхождения x в A.

Определение 1.4. Функция $V(A)$ — множество переменных, входящих в A.

Определение 1.5. Функция $\mathrm{FV}(A)$ — множество свободных переменных, входящих в A:

$$\mathrm{FV}(A) = \begin{cases} \{x\} & \text{если } A \equiv x \\ \mathrm{FV}(P) \cup \mathrm{FV}(Q) & \text{если } A \equiv PQ \\ \mathrm{FV}(P) \setminus \{x\} & \text{если } A \equiv \lambda x.P \end{cases}$$

λ-выражение можно понимать как функцию. Абстракция — это функция с аргументом, аппликация — это передача аргумента.

Определение 1.6 (α-эквивалентность)**.** $A =_\alpha B$, если имеет место одно из следующих условий:

1. $A \equiv x$, $B \equiv y$ и $x \equiv y$.

2. $A \equiv P_1 Q_1$, $B \equiv P_2 Q_2$ и $P_1 =_\alpha P_2$, $Q_1 =_\alpha Q_2$.

3. $A \equiv \lambda x.P_1$, $B \equiv \lambda y.P_2$ и $P_1[x := t] =_\alpha P_2[y := t]$, где t — новая переменная.

Пример. $\lambda x.\lambda y.xy =_\alpha \lambda y.\lambda x.yx$.

Доказательство.

1. $tz =_\alpha tz$ верно по второму условию.

2. Тогда получаем, что $\lambda y.ty =_\alpha \lambda x.tx$ по третьему условию, так как из предыдущего пункта следует $ty[y := z] =_\alpha tx[x := z]$.

3. Из второго пункта получаем, что $\lambda x.\lambda y.xy =_\alpha \lambda y.\lambda x.yx$ по третьему условию, так как $\lambda y.xy[x := t] =_\alpha \lambda x.yx[y := t]$.

□

Определение 1.7 (β-редекс)**.** β-редекс — выражение вида: $(\lambda x.A)\,B$

Определение 1.8 (β-редукция)**.** $A \to_\beta B$, если имеет место одно из следующих условий:

1. $A \equiv P_1 Q_1$, $B \equiv P_2 Q_2$ и либо $P_1 =_\alpha P_2$, $Q_1 \to_\beta Q_2$, либо $P_1 \to_\beta P_2$, $Q_1 =_\alpha Q_2$

2. $A \equiv (\lambda x.P)\,Q$, $B \equiv P[x := Q]$ причем Q свободна для подстановки вместо x в P

3. $A \equiv \lambda x.P$, $B \equiv \lambda x.Q$ и $P \to_\beta Q$

Пример. $(\lambda x.x)\,y \to_\beta y$

Пример. $a\,((\lambda x.x)\,y) \to_\beta ay$

1.2 Представление некоторых функций в λ-исчислении

Логические значения легко представить в терминах λ-исчисления. В самом деле, положим:

- $\text{True} \equiv \lambda a \lambda b.a$
- $\text{False} \equiv \lambda a \lambda b.b$

Также мы можем выражать и более сложные функции

Определение 1.9. $\text{If} \equiv \lambda c.\lambda t.\lambda e.(ct)e$

Пример. $\text{If T } a\ b \to_\beta\ a$

Доказательство.

$$((\lambda c.\lambda t.\lambda e.(ct)e)\ \lambda a\lambda b.a)\ a\ b \to_\beta (\lambda t.\lambda e.(\lambda a\lambda b.a)\ t\ e)\ a\ b \to_\beta$$
$$(\lambda t.\lambda e.(\lambda b.t)\ e)\ a\ b \to_\beta\ (\lambda t.\lambda e.t)\ a\ b \to_\beta\ (\lambda e.a)\ b \to_\beta\ a$$

□

Как мы видим, If T действительно возвращает результат первой ветки.
Другие логические операции:

$$\text{Not} = \lambda a.a\ \text{ F T} \qquad \text{And} = \lambda a.\lambda b.a\ b\ \text{ F} \qquad \text{Or} = \lambda a.\lambda b.a\ \text{ T}\ b$$

1.3 Черчевские нумералы

Определение 1.10 (черчевский нумерал)**.**

$$\overline{n} = \lambda f.\lambda x.f^n x, \quad \text{где} \quad f^n x = \begin{cases} f\,(f^{n-1}x) & \text{при } n > 0 \\ x & \text{при } n = 0 \end{cases}.$$

Пример.

$$\overline{3} = \lambda f.\lambda x.f\,(f\,(fx))$$

Несложно определить прибавление единицы к такому нумералу:

$$(+1) = \lambda n.\lambda f.\lambda x.f(nfx)$$

Арифметические операции:

1. $\mathrm{IsZero} = \lambda n.n\,(\lambda x.\,\mathrm{F})\,\mathrm{T}$
2. $\mathrm{Add} = \lambda a.\lambda b.\lambda f.\lambda x.a\,f\,(b\,f\,x)$
3. $\mathrm{Pow} = \lambda a.\lambda b.b\,(\mathrm{Mul}\ \ a)\,\overline{1}$
4. $\mathrm{IsEven} = \lambda n.n\ \ \mathrm{Not}\ \ \mathrm{T}$
5. $\mathrm{Mul} = \lambda a.\lambda b.a\,(\mathrm{Add}\ \ b)\,\overline{0}$

Для того, чтобы определить (-1), сначала определим пару:

$$\langle a, b\rangle = \lambda f.f\,a\,b \qquad \mathrm{First} = \lambda p.p\,\mathrm{T} \qquad \mathrm{Second} = \lambda p.p\,\mathrm{F}$$

Затем n раз применим функцию $f\,(\langle a, b\rangle) = \langle b, b+1\rangle$ и возьмём первый элемент пары:

$$(-1) = \lambda n.\,\mathrm{First}(n\,(\lambda p.\,\langle(\mathrm{Second}\,p)\,,(+1)\,(\mathrm{Second}\,p)\rangle)\,\langle\overline{0},\overline{0}\rangle)$$

2 Лекция 2

2.1 Формализация λ-термов, классы α-эквивалентности термов

Определение 2.1 (λ-терм)**.** Рассмотрим классы эквивалентности $[A]_{=_\alpha}$.
Будем говорить, что $[A] \to_\beta [B]$, если существуют $A' \in [A]$ и $B' \in [B]$, что $A' \to_\beta B'$.

Лемма 2.1. $(=_\alpha)$ — отношение эквивалентности.

Пусть в A есть β-редекс $(\lambda x.P)Q$, но Q не свободен для подстановки вместо x в P, тогда найдем $y \notin V[P]$, $y \notin V[Q]$. Сделаем замену $P[x := y]$. Тогда замена $P[x := y][y := Q]$ допустима. То есть, можно сказать, что мы просто переименовали переменную x в P и получили свободу для подстановки, тем самым получив возможность редукции.

Лемма 2.2. $P[x := Q] =_\alpha P[x := y][y := Q]$, если замена допустима.

2.2 Нормальная форма, λ-выражения без нормальной формы, комбинаторы K, I, Ω

Определение 2.2. λ-выражение A находится в нормальной форме, если оно не содержит β-редексов.

Определение 2.3. A — нормальная форма B, если существует последовательность термов $A_1 \dots A_n$ такая, что $B =_\alpha A_1 \to_\beta A_2 \to_\beta ... \to_\beta A_n =_\alpha A$ и A находится в нормальной форме.

Определение 2.4. Комбинатор — λ-выражение без свободных переменных.

Определение 2.5.

- $I \equiv \lambda x.x$ (Identitant)
- $K \equiv \lambda a.\lambda b.a$ (Konstanz)
- $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$

Лемма 2.3. Ω — не имеет нормальной формы.

Доказательство. Ω имеет единственный β-редекс, где $A \equiv xx$, $B \equiv (\lambda x.xx)$. Тогда единственный возможный путь редукции — подставить B вместо x в A. Но тогда мы получим Ω. Следовательно, у Ω нет нормальной формы, так как в полученном выражении у нас всегда будет β-редекс. □

2.3 β-редуцируемость

Определение 2.6. Будем говорить, что $A \twoheadrightarrow_\beta B$, если $\exists$ такие $X_1 \dots X_n$, что $A =_\alpha X_1 \to_\beta X_2 \to_\beta \dots \to_\beta X_{n-1} \to_\beta X_n =_\alpha B$.

$(\twoheadrightarrow_\beta)$ — рефлексивное и транзитивное замыкание $(\to_\beta)$. $(\twoheadrightarrow_\beta)$ не обязательно приводит к нормальной форме

Пример. $\Omega \twoheadrightarrow_\beta \Omega$

2.4 Ромбовидное свойство

Определение 2.7 (Ромбовидное свойство)**.** Отношение R обладает ромбовидным свойством, если для любых a, b, c таких, что aRb, aRc, $b \neq c$, существует d, что bRd и cRd.

Пример. $(\leqslant)$ на множестве натуральных чисел обладает ромбовидным свойством, $(>)$ на множестве натуральных чисел не обладает ромбовидным свойством.

2.5 Теорема Чёрча-Россера, следствие о единственности нормальной формы

Теорема 2.4 (Черча-Россера)**.** $(\twoheadrightarrow_\beta)$ обладает ромбовидным свойством.

Следствие 2.1. Если у A есть нормальная форма, то она единственная с точностью до $(=_\alpha)$ (переименования переменных).

Доказательство. Пусть $A \twoheadrightarrow_\beta B$ и $A \twoheadrightarrow_\beta C$. B, C — нормальные формы и $B \neq_\alpha C$. Тогда по теореме Черча-Россера $\exists D$: $B \twoheadrightarrow_\beta D$ и $C \twoheadrightarrow_\beta D$. Тогда $B =_\alpha D$ и $C =_\alpha D \Rightarrow B =_\alpha C$. Противоречие. □

Лемма 2.5. Если B — нормальная форма, то не существует Q такой, что $B \to_\beta Q$. Значит если $B \twoheadrightarrow_\beta Q$, то количество шагов редукции равно 0.

Лемма 2.6. Если R — обладает ромбовидным свойством, то и R^* (транзитивное, рефлексивное замыкание R) им обладает.

Доказательство. Пусть $M_1 R^* M_n$ и $M_1 R N_1$. Тогда существуют такие $M_2 \dots M_{n-1}$, что $M_1 R M_2 \ \dots M_{n-1} R M_n$. Так как R обладает ромбовидным свойством, $M_1 R M_2$ и $M_1 R N_1$, то существует такое N_2, что $N_1 R N_2$ и $M_2 R N_2$. Аналогично, существуют такие $N_3 \dots N_n$, что $N_{i-1} R N_i$ и $M_i R N_i$. Мы получили такое N_n, что $N_1 R^* N_n$ и $M_n R^* N_n$.

Пусть теперь $M_{1,1} R^* M_{1,n}$ и $M_{1,1} R^* M_{m,1}$, то есть имеются $M_{1,2} \dots M_{1,n-1}$ и $M_{2,1} \dots M_{m-1,1}$, что $M_{1,i-1} R M_{1,i}$ и $M_{i-1,1} R M_{i,1}$. Тогда существует такое $M_{2,n}$, что $M_{2,1} R^* M_{2,n}$ и $M_{1,n} R^* M_{2,n}$. Аналогично, существуют такие $M_{3,n} \dots M_{m,n}$, что $M_{i,1} R^* M_{i,n}$ и $M_{1,n} R^* M_{i,n}$. Тогда $M_{1,n} R^* M_{m,n}$ и $M_{m,1} R^* M_{m,n}$. □

Лемма 2.7 (Грустная лемма)**.** $(\to_\beta)$ не обладает ромбовидным свойством.

Доказательство. Пусть $A = (\lambda x.xx)(\mathcal{I}\mathcal{I})$. Покажем, что в таком случае не будет выполняться ромбовидное свойство:

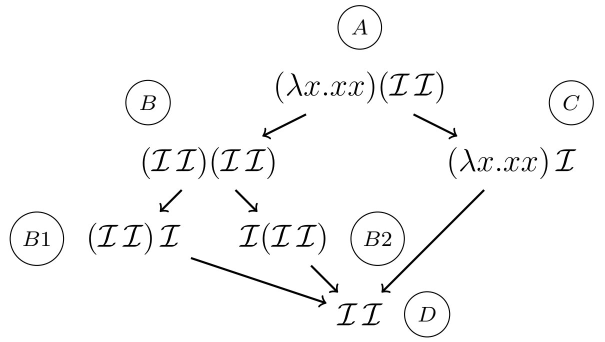


Рис. 1: Нет такого D, что $B \to_\beta D$ и $C \to_\beta D$.

□

Определение 2.8 (Параллельная β-редукция)**.** $A \rightrightarrows_\beta B$, если

1. $A =_\alpha B$
2. $A \equiv P_1 Q_1$, $B \equiv P_2 Q_2$ и $P_1 \rightrightarrows_\beta P_2$, $Q_1 \rightrightarrows_\beta Q_2$
3. $A \equiv \lambda x.P_1$, $B \equiv \lambda x.P_2$ и $P_1 \rightrightarrows_\beta P_2$
4. $A =_\alpha (\lambda x.P_1)Q_1$, $B =_\alpha P_2[x := Q_2]$ причем Q_2 свободна для подстановки вместо x в P_2 и $P_1 \rightrightarrows_\beta P_2$, $Q_1 \rightrightarrows_\beta Q_2$

Лемма 2.8. Если $P_1 \rightrightarrows_\beta P_2$ и $Q_1 \rightrightarrows_\beta Q_2$, то $P_1[x := Q_1] \rightrightarrows_\beta P_2[x := Q_2]$

Доказательство. Будем доказывать индукцией по определению $\rightrightarrows_\beta$. Рассмотрим случаи:

- Пусть $P_1 =_\alpha P_2$. Тогда лемма легко доказывается индукцией по структуре выражения.
- Пусть $P_1 \equiv A_1 B_1$, $P_2 \equiv A_2 B_2$. По определению $(\rightrightarrows_\beta)$ $A_1 \rightrightarrows_\beta A_2$ и $B_1 \rightrightarrows_\beta B_2$. Рассмотрим два случая:

1. $x \in \mathrm{FV}(A_1)$. По индукционному предположению $A_1[x := Q_1] \rightrightarrows_\beta A_2[x := Q_2]$. Тогда $A_1[x := Q_1]B_1 \rightrightarrows_\beta A_2[x := Q_2]B_2$. Тогда $A_1B_1[x := Q_1] \rightrightarrows_\beta A_2B_2[x := Q_2]$.
2. $x \in \mathrm{FV}(B_1)$. По индукционному предположению $B_1[x := Q_1] \rightrightarrows_\beta B_2[x := Q_2]$. Тогда $A_1B_1[x := Q_1] \rightrightarrows_\beta A_2B_2[x := Q_2]$.

- Пусть $P_1 \equiv \lambda y.A_1$, $P_2 \equiv \lambda y.A_2$. По определению $(\rightrightarrows_\beta)$ $A_1 \rightrightarrows_\beta A_2$. Тогда по индукционному предположению $A_1[x := Q_1] \rightrightarrows_\beta A_2[x := Q_2]$. Тогда $\lambda y.(A_1[x := Q_1]) \rightrightarrows_\beta \lambda y.(A_2[x := Q_2])$ по определению $(\rightrightarrows_\beta)$. Следовательно $\lambda y.A_1[x := Q_1] \rightrightarrows_\beta \lambda y.A_2[x := Q_2]$ по определению подстановки.

- Пусть $P_1 =_\alpha (\lambda y.A_1)B_1$, $P_2 =_\alpha A_2[y := B_2]$ и $A_1 \rightrightarrows_\beta A_2$, $B_1 \rightrightarrows_\beta B_2$. По индукционному предположению получаем, что $A_1[x := Q_1] \rightrightarrows_\beta A_2[x := Q_2]$, $B_1[x := Q_1] \rightrightarrows_\beta B_2[x := Q_2]$. Следовательно, по определению $(\rightrightarrows_\beta)$ получаем, что $(\lambda y.A_1[x := Q_1])B_1[x := Q_1] \rightrightarrows_\beta A_2[y := B_2][x := Q_2]$

□

Лемма 2.9. $(\rightrightarrows_\beta)$ обладает ромбовидным свойством.

Доказательство. Будем доказывать индукцией по определению $(\rightrightarrows_\beta)$. Покажем, что если $M \rightrightarrows_\beta M_1$ и $M \rightrightarrows_\beta M_2$, то существует M_3, что $M_1 \rightrightarrows_\beta M_3$ и $M_2 \rightrightarrows_\beta M_3$. Рассмотрим случаи:

- Если $M \equiv M_1$, то просто возьмем $M_3 \equiv M_2$.

- Если $M \equiv \lambda x.P, M_1 \equiv \lambda x.P_1, M_2 \equiv \lambda x.P_2$ и $P \rightrightarrows_\beta P_1, P \rightrightarrows_\beta P_2$, то по предположению индукции существует P_3, что $P_1 \rightrightarrows_\beta P_3, P_2 \rightrightarrows_\beta P_3$, тогда возьмем $M_3 \equiv \lambda x.P_3$.

- Если $M \equiv PQ, M_1 \equiv P_1Q_1$ и по определению $(\rightrightarrows_\beta)$ $P \rightrightarrows_\beta P_1, Q \rightrightarrows_\beta Q_1$, то рассмотрим два случая:

  1. $M_2 \equiv P_2Q_2$. Тогда по предположению индукции существует P_3, что $P_1 \rightrightarrows_\beta P_3, P_2 \rightrightarrows_\beta P_3$. Аналогично для Q. Тогда возьмем $M_3 \equiv P_3Q_3$.
  2. $P \equiv \lambda x.P'$ значит $P_1 \equiv \lambda x.P_1'$ и $P' \rightrightarrows_\beta P_1'$. Пусть тогда $M_2 \equiv P_2[x := Q_2]$, по определению $(\rightrightarrows_\beta)$ $P' \rightrightarrows_\beta P_2, Q \rightrightarrows_\beta Q_2$. Тогда по предположению индукции и лемме 2.8 существует $M_3 \equiv P_3[x := Q_3]$ такой, что $P_1' \rightrightarrows_\beta P_3$, $Q_1 \rightrightarrows_\beta Q_3$ и $P_2 \rightrightarrows_\beta P_3$, $Q_2 \rightrightarrows_\beta Q_3$.

- Если $M \equiv (\lambda x.P)Q$, $M_1 \equiv P_1[x := Q_1]$ и $P \rightrightarrows_\beta P_1$, $Q \rightrightarrows_\beta Q_1$, то рассмотрим случаи:

  1. $M_2 \equiv (\lambda x.P_2)Q_2$, $P \rightrightarrows_\beta P_2$, $Q \rightrightarrows_\beta Q_2$. Тогда по предположению индукции и лемме 2.8 существует такой $M_3 \equiv P_3[x := Q_3]$, что $P_1 \rightrightarrows_\beta P_3$, $Q_1 \rightrightarrows_\beta Q_3$ и $P_2 \rightrightarrows_\beta P_3$, $Q_2 \rightrightarrows_\beta Q_3$.
  2. $M_2 \equiv P_2[x := Q_2]$, $P \rightrightarrows_\beta P_2$, $Q \rightrightarrows_\beta Q_2$. Тогда по предположению индукции и лемме 2.8 существует такой $M_3 \equiv P_3[x := Q_3]$, что $P_1 \rightrightarrows_\beta P_3$, $Q_1 \rightrightarrows_\beta Q_3$ и $P_2 \rightrightarrows_\beta P_3$, $Q_2 \rightrightarrows_\beta Q_3$.

□

Лемма 2.10.

1. $(\rightrightarrows_\beta)^* \subseteq (\rightarrow_\beta)^*$
2. $(\rightarrow_\beta)^* \subseteq (\rightrightarrows_\beta)^*$

Следствие 2.2. $(\rightarrow_\beta)^* = (\rightrightarrows_\beta)^*$

Из приведенных выше лемм и следствия докажем теорему Черча-Россера.

Доказательство. $(\to_\beta)^* = (\twoheadrightarrow_\beta)$. Тогда $(\twoheadrightarrow_\beta) = (\rightrightarrows_\beta)^*$. Значит из того, что $(\rightrightarrows_\beta)$ обладает ромбовидным свойством и леммы 2.6, следует, что $(\twoheadrightarrow_\beta)$ обладает ромбовидным свойством. □

2.6 Нормальный и аппликативный порядок вычислений

Пример. Выражение $KI\Omega$ можно редуцировать двумя способами:

1. $\mathcal{K}\mathcal{I}\,\Omega =_\alpha ((\lambda a.\lambda b.a)\,\mathcal{I})\Omega \to_\beta (\lambda b.\,\mathcal{I})\Omega \to_\beta \mathcal{I}$
2. $\mathcal{K}\mathcal{I}\,\Omega =_\alpha ((\lambda a.\lambda b.a)\,\mathcal{I})((\lambda x.x\ x)(\lambda x.x\ x)) \twoheadrightarrow_\beta ((\lambda a.\lambda b.a)\,\mathcal{I})((\lambda x.x\ x)(\lambda x.x\ x)) \to_\beta \mathcal{K}\mathcal{I}\,\Omega$

Как мы видим, в первом случае мы достигли нормальной формы, в то время как во втором мы получили бесконечную редукцию. Разница двух этих способов в порядке редукции. Первый называется нормальный порядок, а второй аппликативный.

Определение 2.9 (нормальный порядок редукции)**.** Редукция самого левого β-редекса.

Определение 2.10 (аппликативный порядок редукции)**.** Редукция самого левого β-редекса из самых вложенных.

Теорема 2.11 (Приводится без доказательства)**.** Если нормальная форма существует, она может быть достигнута нормальным порядком редукции.

Нормальный порядок хоть и приводит к нормальной форме, если она существует, но бывают ситуации, в которых аппликативный порядок вычисляется быстрее, чем нормальный.

Пример. Рассмотрим λ-выражение $(\lambda x.x\ x\ x\ x)(\mathcal{I}\mathcal{I})$. Попробуем редуцировать его нормальным порядком:

$$(\lambda x.x\,x\,x\,x)(\mathcal{I}\mathcal{I}) \to_\beta (\mathcal{I}\mathcal{I})(\mathcal{I}\mathcal{I})(\mathcal{I}\mathcal{I})(\mathcal{I}\mathcal{I}) \to_\beta \mathcal{I}(\mathcal{I}\mathcal{I})(\mathcal{I}\mathcal{I})(\mathcal{I}\mathcal{I}) \to_\beta (\mathcal{I}\mathcal{I})(\mathcal{I}\mathcal{I})(\mathcal{I}\mathcal{I}) \to_\beta \ldots \to_\beta \mathcal{I}$$

Как мы увидим, в данной ситуации аппликативный порядок редукции оказывается значительно эффективней:

$$(\lambda x.x\ x\ x\ x)(\mathcal{I}\mathcal{I}) \to_\beta (\lambda x.x\ x\ x\ x)\mathcal{I} \to_\beta \mathcal{I}\mathcal{I}\mathcal{I}\mathcal{I} \to_\beta \mathcal{I}\mathcal{I}\mathcal{I} \to_\beta \mathcal{I}\mathcal{I} \to_\beta \mathcal{I}$$

3 Лекция 3

3.1 Y-комбинатор

Определение 3.1. Комбинатором называется λ-выражение, не имеющее свободных переменных

Определение 3.2. (Y-комбинатор)

$$Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

Очевидно, Y-комбинатор является комбинатором.

Теорема 3.1. $Yf =_\beta f(Yf)$

Доказательство. β-редуцируем выражение Yf

$$\begin{aligned} &=_\beta (\lambda f.(\lambda x.f(xx))(\lambda x.f(xx)))f \\ &=_\beta (\lambda x.f(xx))(\lambda x.f(xx)) \\ &=_\beta f((\lambda x.f(xx))(\lambda x.f(xx))) \\ &=_\beta f(Yf) \end{aligned}$$

Так как при второй редукции мы получили, что $Yf =_\beta (\lambda x.f(xx))(\lambda x.f(xx))$ □

Следствием этого утверждения является теорема о неподвижной точке для бестипового λ-исчисления

Теорема 3.2. В λ-исчислении каждый терм f имеет неподвижную точку, то есть такое p, что $f\ p =_\beta p$

Доказательство. Возьмём в качестве p терм Yf. По предыдущей теореме, $f(Yf) =_\beta Yf$, то есть Yf является неподвижной точкой для f. Для любого терма f существует терм Yf, значит, у любого терма есть неподвижная точка. □

3.2 Рекурсия

С помощью Y-комбинатора можно определять рекурсивные функции, например, функцию, вычисляющую факториал Чёрчевского нумерала. Для этого определим вспомогательную функцию

$fact' \equiv \lambda f.\lambda n.isZero\, n\, \overline{1}(mul\ n\ f((-1)n))$

Тогда $fact \equiv Y fact'$

Заметим, что $fact\ \overline{n} =_\beta fact'\ (Y\ fact')\ \overline{n} =_\beta fact'\ fact\ \overline{n}$, то есть в тело функции $fact'$ вместо функции f будет подставлена $fact$ (заметим, что это значит, что именно функция $fact$ будет применена к $\overline{n-1}$, то есть это соответствует нашим представлениям о рекурсии).

Для понимания того, как это работает, посчитаем $fact\ \overline{2}$

$$\begin{aligned} & fact\ \overline{2} \\ &=_\beta Y\ fact'\ \overline{2} \\ &=_\beta fact'(Y\ fact')\overline{2} \\ &=_\beta (\lambda f.\lambda n.isZero\, n\, \overline{1}(mul\ n\ f((-1)n)))(Y\ fact')\overline{2} \\ &=_\beta isZero\, \overline{2}\, \overline{1}(mul\ \overline{2}\ ((Y\ fact')((-1)\overline{2}))) \\ &=_\beta mul\ \overline{2}\ ((Y\ fact')((-1)\overline{2})) \\ &=_\beta mul\ \overline{2}\ (Y\ fact'\ \overline{1}) \\ &=_\beta mul\ \overline{2}\ (fact'\ (Y\ fact'\ \overline{1})) \end{aligned}$$

Раскрывая $fact'\ (Y\ fact'\ \overline{1})$ так же, как мы раскрывали $fact'\ (Y\ fact'\ \overline{2})$, получаем

$$=_\beta mul\ \overline{2}\ (mul\ \overline{1}\ (Y\ fact'\ \overline{0}))$$

Посчитаем $(Y\ fact'\ \overline{0})$

$$
\begin{aligned}
&(Y\ fact'\ \overline{0})\\
&=_\beta fact'\ (Y\ fact')\ \overline{0}\\
&=_\beta (\lambda f.\lambda n.isZero\ n\ \overline{1}(mul\ n\ f((-1)n)))\ (Y\ fact')\ \overline{0}\\
&=_\beta isZero\ \overline{0}\ \overline{1}(mul\ \overline{0}\ ((Y\ fact'))((-1)\overline{0})) =_\beta \overline{1}
\end{aligned}
$$

Таким образом,

$$
\begin{aligned}
&fact\ \overline{2}\\
&=_\beta mul\ \overline{2}\ (mul\ \overline{1}\ (Y\ fact'\ \overline{0}))\\
&=_\beta mul\ \overline{2}\ (mul\ \overline{1}\ \overline{1}) =_\beta mul\ \overline{2}\ \overline{1} =_\beta \overline{2}
\end{aligned}
$$

3.3 Парадокс Карри

Попробуем построить логику на основе λ-исчисления. Введём логический символ $\to$.

Будем требовать от этого исчисления наличия следующих схем аксиом:

1. $\vdash A \to A$
2. $\vdash (A \to (A \to B)) \to (A \to B)$
3. $\vdash A =_\beta B$, тогда $A \to B$

А также правила вывода MP:

$$\frac{\vdash A \to B,\ \vdash A}{\vdash B}$$

Не вводя дополнительные правила вывода и схемы аксиом, покажем, что данная логика является противоречивой. Для чего введём следующие условные обозначения:

$F_\alpha \equiv \lambda x.(x\ x) \to \alpha$

$\Phi_\alpha \equiv F_\alpha\ F_\alpha \equiv (\lambda x.(x\ x) \to \alpha)\ (\lambda x.(x\ x) \to \alpha)$

Редуцируя Φ_α, получаем

$$
\begin{aligned}
&\Phi_\alpha\\
&=_\beta (\lambda x.(x\ x) \to \alpha)\ (\lambda x.(x\ x) \to \alpha)\\
&=_\beta (\lambda x.(x\ x) \to \alpha)\ (\lambda x.(x\ x) \to \alpha) \to \alpha\\
&=_\beta \Phi_\alpha \to \alpha
\end{aligned}
$$

Теперь докажем противоречивость введённой логики. Для этого докажем, что в ней выводимо любое утверждение.

1) $\vdash \Phi_\alpha \to \Phi_\alpha \to \alpha$	Так как $\Phi_\alpha =_\beta \Phi_\alpha \to \alpha$
2) $\vdash (\Phi_\alpha \to \Phi_\alpha \to \alpha) \to (\Phi_\alpha \to \alpha)$	Так как $\vdash (A \to (A \to B)) \to (A \to B)$
3) $\vdash \Phi_\alpha \to \alpha$	MP 2, 3
4) $\vdash (\Phi_\alpha \to \alpha) \to \Phi_\alpha$	Так как $\vdash \Phi_\alpha \to \alpha =_\beta \Phi_\alpha$
5) $\vdash \Phi_\alpha$	MP 3, 4
6) $\vdash \alpha$	MP 3, 5

Таким образом, введённая логика оказывается противоречивой.

3.4 Импликационный фрагмент интуиционистского исчисления высказываний

Рассмотрим подмножество ИИВ, со следующей грамматикой:

$\Phi ::= x \mid \Phi \to \Phi \mid (\Phi)$

То есть состоящее только из переменных и импликаций.

Добавим в него одну схему аксиом

$$\Gamma, \varphi \vdash \varphi$$

И два правила вывода

1. Правило введения импликации:
$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}$$

2. Правило удаления импликации:
$$\frac{\Gamma \vdash \varphi \to \psi \qquad \Gamma \vdash \varphi}{\Gamma \vdash \psi}$$

Пример. Докажем $\vdash \varphi \to \psi \to \varphi$

$$\cfrac{\cfrac{\varphi, \psi \vdash \varphi}{\varphi \vdash \psi \to \varphi}\text{ (Введение импликации)}}{\vdash \varphi \to (\psi \to \varphi)}\text{ (Введение импликации)}$$

Пример. Докажем $\alpha \to \beta \to \gamma,\ \alpha,\ \beta \vdash \gamma$

$$\cfrac{\cfrac{\alpha \to \beta \to \gamma,\ \alpha,\ \beta \vdash \alpha \to \beta \to \gamma \qquad \alpha \to \beta \to \gamma,\ \alpha,\ \beta \vdash \alpha}{\alpha \to \beta \to \gamma,\ \alpha,\ \beta \vdash \beta \to \gamma} \qquad \alpha \to \beta \to \gamma, \alpha,\ \beta \vdash \beta}{\alpha \to \beta \to \gamma,\ \alpha,\ \beta \vdash \gamma}$$

3.5 Просто типизированное по Карри λ-исчисление

Определение 3.3. Тип в просто типизированном λ-исчислении по Карри — это либо маленькая греческая буква $(\alpha, \phi, \theta, \ldots)$, либо импликация $(\theta_1 \to \theta_2)$

Таким образом, $\Theta ::= \theta_i | \Theta \to \Theta | (\Theta)$

Импликация при этом считается правоассоциативной операцией.

Определение 3.4. Язык просто типизированного λ-исчисления — это язык бестипового λ-исчисления.

Определение 3.5. Контекст Γ — это список выражений вида $A : \theta$, где A — λ-терм, а θ — тип.

Определение 3.6. Просто типизированное λ-исчисление по Карри.

Рассмотрим исчисление с единственной схемой аксиом:

$$\Gamma, x : \theta \vdash x : \theta, \text{если } x \text{ не входит в } \Gamma$$

И следующими правилами вывода

1. Правило типизации абстракции

$$\frac{\Gamma, x : \varphi \vdash P : \psi}{\Gamma \vdash (\lambda\, x.\, P) : \varphi \to \psi} \text{ если } x \text{ не входит в } \Gamma$$

2. Правило типизации аппликации:

$$\frac{\Gamma \vdash P : \varphi \to \psi \qquad \Gamma \vdash Q : \varphi}{\Gamma \vdash PQ : \psi}$$

Если λ-выражение типизируется с использованием этих двух правил и одной схемы аксиом, то будем говорить, что оно типизируется по Карри.

Пример. Докажем $\vdash \lambda\, x.\, \lambda\, y.\, x : \alpha \to \beta \to \alpha$

$$\cfrac{\cfrac{x : \alpha, y : \beta \vdash x : \alpha}{x : \alpha \vdash \lambda\, y.\, x : \beta \to \alpha} \text{(Правило типизации абстракции)}}{\vdash \lambda\, x.\, \lambda\, y.\, x : \alpha \to \beta \to \alpha} \text{(Правило типизации абстракции)}$$

Пример. Докажем $\vdash \lambda\, x.\, \lambda\, y.\, x\, y : (\alpha \to \beta) \to \alpha \to \beta$

$$\cfrac{\cfrac{\cfrac{x : \alpha \to \beta, y : \alpha \vdash x : \alpha \to \beta \qquad x : \alpha \to \beta, y : \alpha \vdash y : \alpha}{x : \alpha \to \beta, y : \alpha \vdash x\, y : \beta}}{x : \alpha \to \beta \vdash \lambda\, y.\, x\, y : \alpha \to \beta}}{\vdash \lambda\, x.\, \lambda\, y.\, x\, y : (\alpha \to \beta) \to \alpha \to \beta}$$

3.6 Отсутствие типа у Y-комбинатора

Теорема 3.3. Y-комбинатор не типизируется в просто типизированном по Карри λ-исчислении.

Неформальное доказательство $Y\ f =_\beta f\ (Y\ f)$, поэтому $Y\ f$ и $f\ (Y\ f)$ должны иметь одинаковые типы.

Пусть $Y\ f : \alpha$

Тогда $Y : \beta \to \alpha, f : \beta$

Из $f\ (Y\ f) : \alpha$ получаем $f : a \to \alpha$ (так как $Yf : \alpha$)

Тогда $\beta = \alpha \to \alpha$, из этого получаем $Y : (\alpha \to \alpha) \to \alpha$

Можно доказать, что $\lambda\ x.\ x : \alpha \to \alpha$. Тогда $Y\ \lambda\ x.\ x : \alpha$, то есть любой тип является обитаемым. Так как это невозможно, Y-комбинатор не может иметь типа, так как тогда он сделает нашу логику противоречивой.

Формальное доказательство Докажем от противного. Пусть Y-комбинатор типизируем. Тогда в выводе его типа есть вывод типа выражения $x\ x$. Так как $x\ x$ — абстракция, то и типизирована она может быть только по правилу абстракции. Значит, в выводе типа Y-комбинатора есть такой вывод:

$$\frac{\Gamma \vdash x : \varphi \to \psi \qquad \Gamma \vdash x : \varphi}{\Gamma \vdash xx : \psi}$$

Рассмотрим типизацию $\Gamma \vdash x : \varphi \to \psi$ и $\Gamma \vdash x : \varphi$. x это атомарная переменная, значит, она могла быть типизирована только по единственной схеме аксиом.

Следовательно, x типизируется следующим образом.

$$\frac{\Gamma', x : \varphi \to \psi, x : \varphi \vdash x : \varphi \to \psi \qquad \Gamma', x : \varphi \to \psi, x : \varphi \vdash x : \varphi}{\Gamma', x : \varphi \to \psi, x : \varphi \vdash xx : \psi}$$

Следовательно, в контексте Γ переменная x встречается два раза, что невозможно по схеме аксиом.

3.7 Изоморфизм Карри-Ховарда

Заметим, что аксиомы и правила вывода импликационного фрагмента ИИВ и просто типизированного по Карри λ-исчисления точно соответствуют друг другу.

Просто типизированное λ-исчисление	Импликативный фрагмент ИИВ
$\Gamma, x : \theta \vdash x : \theta$	$\Gamma, \varphi \vdash \varphi$
$\dfrac{\Gamma, x : \varphi \vdash P : \psi}{\Gamma \vdash (\lambda\, x.\, P) : \varphi \to \psi}$	$\dfrac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}$
$\dfrac{\Gamma \vdash P : \varphi \to \psi \qquad \Gamma \vdash Q : \varphi}{\Gamma \vdash PQ : \psi}$	$\dfrac{\Gamma \vdash \varphi \to \psi \qquad \Gamma \vdash \varphi}{\Gamma \vdash \psi}$

Установим соответствие и между прочими сущностями ИИВ и просто типизированного по Карри λ-исчисления.

Просто типизированное λ-исчисление	Импликативный фрагмент ИИВ
Тип	Высказывание
Терм	Доказательство высказывания
Проверка того, что терм имеет заданный тип	Проверка доказательства на корректность
Обитаемый тип	Доказуемое высказывание
Проверка того, что существует терм, имеющий заданный тип	Проверка того, что заданное высказывание имеет доказательство

4 Лекция 4

4.1 Расширение просто типизированного λ-исчисления до изоморфного ИИВ

Заметим, что между просто типизированным по Карри λ-исчислением и импликационным фрагментом ИИВ существует изоморфизм, но при этом в просто типизированном λ-исчислении нет аналогов лжи, а также связок $\vee$ и $\&$.

Для установления полного изоморфизма между ИИВ и просто типизированным λ-исчислением введём три необходимые для установления этого изоморфизма сущности:

1. Тип "Ложь"($\perp$)

2. Тип упорядоченной пары $A\&B$, соответствующий логическому "И"

3. Алгебраический тип $A|B$, соответствующий логическому "ИЛИ"

Тип $\perp$ Введём тип $\perp$, соответствующий лжи в ИИВ. Поскольку из лжи может следовать что угодно, добавим в исчисление новое правило вывода

$$\frac{\Gamma \vdash A : \perp}{\Gamma \vdash A : \tau}$$

То есть выражение, типизированное как $\perp$, может быть типизировано также любым другим типом.

В программировании аналогом этого типа может являться тип `Nothing`, который является подтипом любого другого типа.

Тип `Nothing` является необитаемым, им типизируется выражение, никогда не возвращающее свой результат (например, `throw new Error() : Nothing`).

Тот факт, что выражение, типизированное как `Nothing`, может быть типизировано любым другим типом, позволяет писать следующие функции:

```
def assertStringNotEmpty(s: String): String = {
  if (s.length != 0) {
    s
  } else {
    throw new Error("Empty string")
  }
}
```

так как `throw new Error("Empty string"): Nothing`, то

`throw new Error("Empty string"): String`, поэтому функция может иметь тип `String`.

Теперь, имея тип $\perp$, можно ввести связку "Отрицание". Обозначим $\neg A = A \rightarrow \perp$, то есть в программировании это будет соответствовать функции

```
def throwError(a: A): Nothing = throw new Error()
```

Упорядоченные пары Введём возможность запаковывать значения в пары.

Функция $makePair$ будет выглядеть следующим образом:

$$makePair \equiv \lambda\ first.\ \lambda\ second.\ \lambda\ f.\ f\ first\ second$$

Тогда

$$< first, second > \equiv makePair\ first\ second$$

Надо также написать функции, которые будут доставать из пары упакованные в неё значения. Назовём их Π_1 и Π_2.

Пусть

$$\Pi_1 \equiv \lambda\ Pair.\ Pair\ (\lambda\ a.\lambda\ b.\ a)$$

$$\Pi_2 \equiv \lambda\ Pair.\ Pair\ (\lambda\ a.\lambda\ b.\ b)$$

Заметим, что

$$
\begin{aligned}
&\Pi_1 < A, B > \\
&=_\beta (\lambda\ Pair.\ Pair\ (\lambda\ a. \lambda\ b.\ a))(makePair\ A\ B) \\
&=_\beta (\lambda\ Pair.\ Pair\ (\lambda\ a. \lambda\ b.\ a))((\lambda\ first.\ \lambda\ second.\ \lambda\ f.\ f\ first\ second)\ A\ B) \\
&=_\beta (\lambda\ Pair.\ Pair\ (\lambda\ a. \lambda\ b.\ a))((\lambda\ second.\ \lambda\ f.\ f\ A\ second)\ B) \\
&=_\beta (\lambda\ Pair.\ Pair\ (\lambda\ a. \lambda\ b.\ a))(\lambda\ f.\ f\ A\ B) \\
&=_\beta (\lambda\ f.\ f\ A\ B)\ (\lambda\ a. \lambda\ b.\ a) \\
&=_\beta (\lambda\ a. \lambda\ b.\ a)\ A\ B \\
&=_\beta (\lambda\ b.\ A)\ B \\
&=_\beta A
\end{aligned}
$$

Аналогично, $\Pi_2 < A, B > =_\beta B$

Таким образом, мы умеем запаковывать элементы в пары и доставать элементы из пар. Теперь, добавим к просто типизированному λ-исчислению правила вывода, позволяющие типизировать такие конструкции.

Добавим три новых правила вывода:

1. Правило типизации пары
$$\frac{\Gamma \vdash A : \varphi \qquad \Gamma \vdash B : \psi}{\Gamma \vdash < A, B >: \varphi \& \psi}$$

2. Правило типизации первого проектора:
$$\frac{\Gamma \vdash < A, B >: \varphi \& \psi}{\Gamma \vdash \Pi_1 < A, B >: \varphi}$$

3. Правило типизации второго проектора:
$$\frac{\Gamma \vdash < A, B >: \varphi \& \psi}{\Gamma \vdash \Pi_2 < A, B >: \psi}$$

Алгебраические типы Добавим тип, который является аналогом `union` в C++, или алгебраического типа в любом функциональном языке. Это тип, который может содержать одну из двух альтернатив.

Например, тип `OptionInt = None | Some of Int` может содержать либо `None`, либо `Some of Int`, но не обе альтернативы разом, причём в каждый момент времени известно, какую альтернативу он содержит.

Заметим, что определение алгебраического типа похоже на определение дизъюнкции в ИИВ (в ИИВ если выполнено $\vdash a \vee b$, известно, что из $\vdash a$ и $\vdash b$ выполнено).

Для реализации алгебраических типов в λ-исчислении напишем три функции:

1. in_1, создающее экземпляр алгебраического типа из первой альтернативы, то есть запаковывающее первую альтернативу в алгебраический тип

2. in_2, выполняющее аналогичные действия, но со второй альтернативой.

3. $case$, принимающую три параметра: экземпляр алгебраического типа, функцию, определяющую, что делать, если этот экземпляр был создан из первой альтернативы (то есть с использованием in_1), и функцию, определяющую, что делать, если этот экземпляр был создан из второй альтернативы (то есть с использованием in_2)

Аналогом *case* в программировании является конструкция, известная как pattern-matching, или сопоставление с образцом.

```
let isEmptyList list = match list with
        | Nil -> true
        | Cons(_, _) -> false
;;
```

Функция in_1 будет выглядеть следующим образом:

$$in_1 \equiv \lambda\, x.\ \lambda\, f.\ \lambda\, g.\ f\ x$$

А in_2 - следующим:

$$in_2 \equiv \lambda\, x.\ \lambda\, f.\ \lambda\, g.\ g\ x$$

То есть in_1 принимает две функции и применяет первую к x, а in_2 применяет вторую. Тогда *case* будет выглядеть следующим образом:

$$case \equiv \lambda\, algebraic.\ \lambda\, f.\ \lambda\, g.\ algebraic\ f\ g$$

Заметим, что

$$\begin{aligned}
& case\ (in_1 A)\ F\ G \\
=_\beta\ & (\lambda\, algebraic.\ \lambda\, f.\ \lambda\, g.\ algebraic\ f\ g)\ ((\lambda\, x.\ \lambda\, h.\ \lambda\, s.\ h\ x) A)\ F\ G \\
=_\beta\ & (\lambda\, algebraic.\ \lambda\, f.\ \lambda\, g.\ algebraic\ f\ g)\ (\lambda\, h.\ \lambda\, s.\ h\ A)\ F\ G \\
=_\beta\ & (\lambda\, f.\ \lambda\, g.\ (\lambda\, h.\ \lambda\, s.\ h\ A)\ f\ g)\ F\ G \\
=_\beta\ & (\lambda\, g.\ (\lambda\, h.\ \lambda\, s.\ h\ A)\ F\ g)\ G \\
=_\beta\ & (\lambda\, h.\ \lambda\, s.\ h\ A)\ F\ G \\
=_\beta\ & (\lambda\, s.\ F\ A)\ G \\
=_\beta\ & F\ A
\end{aligned}$$

Аналогично, $case\ (in_2 B)\ F\ G =_\beta G\ B$.

То есть $case, in_1$ и in_2 умеют применять нужную функцию к запакованной в экземпляр алгебраического типа одной из альтернатив.

Теперь добавим к просто типизированному λ-исчислению правила вывода, позволяющие типизировать эти конструкции.

Добавим три новых правила вывода:

1. Правило типизации левой инъекции

$$\frac{\Gamma \vdash A : \varphi}{\Gamma \vdash in_1\ A : \varphi \vee \psi}$$

2. Правило типизации правой инъекции:

$$\frac{\Gamma \vdash B : \psi}{\Gamma \vdash in_2\ B : \varphi \vee \psi}$$

3. Правило типизации case:

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

4.2 Изоморфизм Карри-Ховарда для расширения просто типизированного λ-исчисления

Заметим точное соответствие только что введённых конструкций аксиомам ИИВ.

Расширенное просто типизированное λ-исчисление	ИИВ
$\dfrac{\Gamma \vdash A : \varphi \qquad \Gamma \vdash B : \psi}{\Gamma \vdash < A, B >: \varphi \& \psi}$	$\vdash \varphi \to \psi \to \varphi \& \psi$
$\dfrac{\Gamma \vdash < A, B >: \varphi \& \psi}{\Gamma \vdash \Pi_1 < A, B >: \varphi}$	$\vdash \varphi \& \psi \to \varphi$
$\dfrac{\Gamma \vdash < A, B >: \varphi \& \psi}{\Gamma \vdash \Pi_2 < A, B >: \psi}$	$\vdash \varphi \& \psi \to \psi$
$\dfrac{\Gamma \vdash A : \varphi}{\Gamma \vdash in_1\ A : \varphi \vee \psi}$	$\vdash \varphi \to \varphi \vee \psi$
$\dfrac{\Gamma \vdash B : \psi}{\Gamma \vdash in_2\ B : \varphi \vee \psi}$	$\vdash \psi \to \varphi \vee \psi$
$\dfrac{\Gamma \vdash L : \varphi \vee \psi,\ \ \Gamma \vdash f : \varphi \to \tau,\ \ \Gamma \vdash g : \psi \to \tau}{case\ L\ f\ g : \tau}$	$\vdash (\varphi \to \tau) \to (\psi \to \tau) \to (\varphi \vee \psi) \to \tau$

4.3 Просто типизированное по Чёрчу λ-исчисление

Определение 4.1. Тип в просто типизированном по Чёрчу λ-исчислении — это то же самое, что тип в просто типизированном по Карри λ-исчислении

Определение 4.2. Язык просто типизированного по Чёрчу λ-исчисления удовлетворяет следующей грамматике
$\Lambda_{\text{ч}} ::= x \mid \Lambda_{\text{ч}}\ \Lambda_{\text{ч}} \mid \lambda\ x^{\tau}.\ \Lambda_{\text{ч}} \mid (\Lambda_{\text{ч}})$

Замечание 4.1. Иногда абстракция записывается не как $\lambda\ x^{\tau}.\ \Lambda_{\text{ч}}$, а как $\lambda\ x : \tau.\ \Lambda_{\text{ч}}$

Определение 4.3. Просто типизированное по Чёрчу λ-исчисление.

Рассмотрим исчисление с единственной схемой аксиом:

$$\Gamma, x : \theta \vdash x : \theta, \text{если } x \text{ не входит в } \Gamma$$

И следующими правилами вывода

1. Правило типизации абстракции

$$\frac{\Gamma, x : \varphi \vdash P : \psi}{\Gamma \vdash (\lambda\ x : \varphi.\ P) : \varphi \to \psi} \text{ если } x \text{ не входит в } \Gamma$$

2. Правило типизации аппликации:

$$\frac{\Gamma \vdash P : \varphi \to \psi \qquad \Gamma \vdash Q : \varphi}{\Gamma \vdash PQ : \psi}$$

Если λ-выражение типизируется с использованием этих двух правил и одной схемы аксиом, то будем говорить, что оно типизируется по Чёрчу.

В исчислении по Чёрчу остаются верными все предыдущие теоремы (в том числе теорема Чёрча-Россера), но правило строгой типизации абстракций позволяет доказать ещё одну теорему:

Теорема 4.1 (Уникальность типов в исчислении по Чёрчу)**.**

1. Если $\Gamma \vdash_{ч} M : \theta$ и $\Gamma \vdash_{ч} M : \tau$, то $\theta = \tau$
2. Если $\Gamma \vdash_{ч} M : \theta$ и $\Gamma \vdash_{ч} N : \tau$, и $M =_\beta N$ то $\theta = \tau$

4.4 Связь типизации по Чёрчу и по Карри

Определение 4.4 (Стирание)**.** Функцией стирания называется следующая функция: $|\cdot| : \Lambda_{ч} \to \Lambda_{к}$:

$$|A| = \begin{cases} x & A \equiv x \\ |M|\,|N| & A \equiv M\ N \\ \lambda x.|P| & A \equiv \lambda\ x : \tau.\ P \end{cases}$$

Лемма 4.2. Пусть $M, N \in \Lambda_{ч}$, $M \to_\beta N$, тогда $|M| \to_\beta |N|$

Лемма 4.3. Если $\Gamma \vdash_{ч} M : \tau$, тогда $\Gamma' \vdash_{к} |M| : \tau$, где Γ' получается из Γ применением функции стирания к каждому терму из Γ

Теорема 4.4 (Теорема о поднятии)**.**

1. Пусть $M, N \in \Lambda_{к}, P \in \Lambda_{ч}, |P| = M, M \to_\beta N$. Тогда найдётся такое $Q \in \Lambda_{ч}$, что $|Q| = N$, и $P \to_\beta Q$
2. Пусть $M \in \Lambda_{к}, \Gamma \vdash_{к} M : \tau$. Тогда существует $P \in \Lambda_{ч}$, что $|P| = M$, и $\Gamma \vdash_{ч} P : \tau$

5 Лекция 5 Изоморфизм Карри-Ховарда (завершение), Унификация

5.1 Изоморфизм Карри-Ховарда

Определение 5.1. Изоморфизм Карри-Ховарда

1. $\Gamma \vdash M : \sigma$ влечет $|\Gamma| \vdash \sigma$ т.е. $|\{x_1 : \Theta_1\ \dots\ x_n : \Theta_n\}| = \{\Theta_1\ \dots\ \Theta_n\}$
2. Если $\Gamma \vdash \sigma$, то существует M и существует Δ, такое что $|\Delta| = \Gamma$, что $\Delta \vdash M : \sigma$, где $\Delta = \{x_\sigma : \sigma \mid \sigma \in \Gamma\}$

Пример. $\{f\ :\ \alpha\ \to\ \beta,\ x\ :\ \beta\}\ \vdash\ f\ x\ :\ \beta$
Применив изоморфизм Карри-Ховарда, получим: $\{\alpha\ \to\ \beta,\ \beta\}\ \vdash\ \beta$

Доказательство. П.1 доказывается индукцией по длине выражения

1. $\Gamma,\ x : \Theta \vdash x : \Theta \qquad \Rightarrow_{KH} \qquad |\Gamma|,\ \Theta \vdash\ \Theta$

2.
$$\frac{\Gamma,\ x : \tau_1 \vdash P : \tau_2}{\Gamma \vdash \lambda x.\, P : \tau_1 \ \to\ \tau_2} \qquad \Rightarrow_{KH} \qquad \frac{|\Gamma|, \tau_1 \vdash \tau_2}{|\Gamma|\ \vdash\ \tau_1\ \to\ \tau_2}$$

3.
$$\frac{\Gamma \vdash P : \tau_1\ \to\ \tau_2 \qquad \Gamma \vdash Q : \tau_1}{\Gamma \vdash P\,Q : \tau_2} \qquad \Rightarrow_{KH} \qquad \frac{|\Gamma| \vdash \tau_1\ \to\ \tau_2 \qquad |\Gamma| \vdash \tau_1}{|\Gamma| \vdash \tau_2}$$

П.2 доказывается аналогичным способом, но действия обратные.
Т.е. отношения между типами в системе типов могут рассматриваться как образ отношений между высказываниями в логической системе, и наоборот. □

Определение 5.2. Расширенный полином:

$$E(p,\ q) = \begin{cases} C, & \text{if } p = q = 0 \\ p_1\,(p), & \text{if } q = 0 \\ p_2\,(q), & \text{if } p = 0 \\ p_3\,(p,\ q), & \text{if } p,\ q \neq 0 \end{cases}$$

где C – константа, $p_1,\ p_2,\ p_3$ – выражения, составленные из $*,\ +,\ p,\ q$ и констант.

Пусть $\upsilon = (\alpha \to \alpha) \to (\alpha \to \alpha)$, где α–произвольный тип и пусть $F \in \Lambda$, что $F : \upsilon \to \upsilon \to \upsilon$, то существует расширенный полином E, такой что $\forall a,\ b \in \mathbb{N}\ F(\overline{a},\ \overline{b}) =_\beta\ \overline{E(a,\ b)}$, где $\overline{a}$–черчевский нумерал.

Теорема 5.1. У каждого терма в просто типизируемом λ исчислении существует расширенный полином.

Утверждение 5.1. Типы черчевских нумералов

1. $0 : \lambda f\ \lambda x.\, x : a\ \to\ b\ \to\ b$
2. $1 : \lambda f\ \lambda x.\, f\ x : (a\ \to\ b)\ \to\ a\ \to\ b$
3. $2 : \lambda f\ \lambda x.\, f\ (f\ x) : (a\ \to\ a)\ \to\ a\ \to\ a$
4. $\forall\, i,\ i \geqslant 2 \quad \lambda f\ \lambda x.\, f(\ \ldots\ (f\ x)) : (a\ \to\ a)\ \to\ a\ \to\ a$

Доказательство. Пункты $1, 2, 3$– очевидно. Рассмотрим более подробно пункт 4:
Разберем нумерал и рассмотрим два последних шага –

$$\cfrac{\cfrac{\cfrac{\cfrac{f : a\ \to\ b \vdash x : a \quad \{1\}}{f : a\ \to\ b \vdash f\ x : b \quad \{2\}}}{f : a\ \to\ b \vdash f\ (f\ x) : \bot \quad \{3\}}}{\ldots}}{\lambda f\ \lambda x.\, f(\ \ldots\ (f\ x))}$$

на шаге 3 становится понятно, что $f : a\ \to\ a$ и $x : a$
($\bot$ в данном контексте означает, что такой терм не типизируем в данном предположении)
□

Утверждение 5.2. Основные задачи типизации λ-исчисления

1. *Проверка типа*–выполняется ли $\Gamma \vdash M : \sigma$ для контекста Γ, терма M и типа σ (для проверки типа обычно опускают σ и рассматривают п.2).

2. *Реконструкция типа*—можно ли подставить вместо ? и $?_1$ в $?_1 \vdash M : ?$ конкретный тип σ в ? и контекст Γ в $?_1$.

3. *Обитаемость типа*—пытается подобрать, такой терм M и контекст Γ, чтобы было выполнено $\Gamma \vdash M : \sigma$.

Определение 5.3. Алгебраический терм

$$\Theta ::= a \mid (f\ \Theta_1\ \ldots\ \Theta_n)$$

где a—переменная, $(f\ \Theta_1\ \ldots\ \Theta_n)$—применение функции

5.2 Уравнение в алгебраических термах $\Theta_1 = \Theta_2$ Система уравнений в алгебраических термах

Определение 5.4. Система уравнений в алгебраических термах

$$\begin{cases} \Theta_1 = \sigma_1 \\ \vdots \\ \Theta_n = \sigma_n \end{cases}$$

где Θ_i и σ_i – термы

Определение 5.5. $\{a_i\} = A$—множество переменных, $\{\Theta_i\} = T$—множество термов.

Определение 5.6. Подстановка—отображение вида: $S_0 : A \to T$, которое является решением в алгебраических термах.

$S_0(a)$ может быть либо $S_0(a) = \Theta_i$, либо $S_0(a) = a$.

S то же, что и много if'ов, либо *map* строк. Доопределим $S : T \to T$, где

1. $S(a) = S_0(a)$

2. $S(f(\Theta_1 \ldots \Theta_k)) = f(S(\Theta_1) \ldots S(\Theta_k))$

Определение 5.7. Решить уравнение в алгебраических термах—найти такое S, что $S(\Theta_1) = S(\Theta_2)$

Пример.

Заранее обозначим: $a,\ b$ – переменные $\quad f,\ g,\ h$ – функции

1. $f\ (a\ (g\ b)) = f\ (h\ e)\ d$ имеет решение $S(a) = h\ e$ и $S(d) = g\ b$

   (a) $S(f\ a\ (g\ b)) = f\ (h\ e)\ (g\ b)$

   (b) $S(f\ (h\ e)\ d) = f\ (h\ e)\ (g\ b)$

   (c) $f\ (h\ e)\ (g\ b) = f\ (h\ e)\ (g\ b)$

2. $f\ a = g\ b$—решений не имеет

Таким образом, чтобы существовало решение, необходимо равенство строк полученной подстановки.

5.3 Алгоритм Унификации. Определения

1. Система уравнений E_1 эквивалентна E_2, если они имеют одинаковые решения(унификаторы).

2. Любая система E эквивалентна некоторому уравнению $\Sigma_1 = \Sigma_2$.

   Доказательство. Возьмем функциональный символ f, не использующийся в E,

   $$E = \begin{cases} \Theta_1 = \sigma_1 \\ \vdots \\ \Theta_n = \sigma_n \end{cases}$$

   это же уравнение можно записать как$-$ $f\ \Theta_1 \ldots \Theta_n = f\ \sigma_1 \ldots \sigma_n$

   Если существует подстановка S такая, что

   $S(\Theta_i) = S(\sigma_i)\ \forall\ i$, то $S(f\ \Theta_1 \ldots \Theta_n) = f\ S(\sigma_1) \ldots S(\sigma_n)$

   Обратное аналогично. □

3. Рассмотрим операции

   (a) *Редукция терма*

   Заменим уравнение вида$-f_1\ \Theta_1 \ldots \Theta_n = f_1\ \sigma_1 \ldots \sigma_n$ на систему уравнений

   $\Theta_1 = \sigma_1$

   $\vdots$

   $\Theta_n = \sigma_n$

   (b) *Устранение переменной*

   Пусть есть уравнение $x = \Theta$, заменим во всех остальных уравнениях переменную x на терм Θ.

   Утверждение 5.3. Эти операции не изменяют множества решений.

   Доказательство. Пункт a – доказан выше, докажем теперь пункт b :

   Пусть есть решение вида $T = \begin{cases} a = \Theta_a \\ \vdots \end{cases}$ и уравнение вида $f\ a\ \ldots\ z\ =\ \Theta_c$, тогда,

   $T(f\ a\ \ldots\ z) = f\ T(a)\ \ldots\ T(z)$, которое в свою очередь является $f\ \Theta_a\ \ldots\ T(z)$ □

Определение 5.8. Система уравнений в разрешенной форме, если

1. Все уравнения имеют вид $a_i = \Theta_i$

2. Каждый из a_i входит в систему уравнений только один раз

Определение 5.9. Система несовместна, если

1. существует уравнение вида $f\ \Theta_1 \ldots \Theta_n = g\ \sigma_1 \ldots \sigma_n$, где $f \neq g$

2. существует уравнение вида $a = f\ \Theta_1 \ldots \Theta_n$, причем a входит в какой-то из Θ_i

5.4 Алгоритм унификации

1. Пройдемся по системе, выберем такое уравнение, что оно удовлетворяет одному из условий:

   (a) Если $\Theta_i = a_i$, то перепишем, как $a_i = \Theta_i$, $\Theta_i -$не переменная

   (b) $a_i = a_i -$ удалим

   (c) $f\ \Theta_1 \dots \Theta_n = f\ \sigma_1 \dots \sigma_n -$ применим редукцию термов

   (d) $a_i = \Theta_i -$Применим подстановку переменной – подставим во все остальные уравнения Θ_i вместо a_i (Если a_i встречается в системе где-то еще)

2. Проверим разрешима ли система, совместна ли система (два пункта несовместимости)

3. Повторим пункт 1

Утверждение 5.4. Алгоритм не изменяет множества решений

Утверждение 5.5. Несовместная система не имеет решений

Утверждение 5.6. Если система имеет решение, то его разрешенная форма единственна

Утверждение 5.7. Система в разрешенной форме имеет решение:

$$\begin{cases} a_1 = \Theta_1 \\ \vdots \\ a_n = \Theta_n \end{cases} \quad \text{имеет решение } - \quad \begin{cases} S_0(a_1) = \Theta_1 \\ \vdots \\ S_0(a_n) = \Theta_n \end{cases}$$

Утверждение 5.8. Алгоритм всегда заканчивается

Доказательство. По индукции, выберем три числа $\langle x\ y\ z \rangle$, где

$x -$количество переменных, которые встречаются строго больше одного раза в левой части некоторого уравнения (b не повлияет на x, а a повлияет в уравнении $f(a(g\,a)\,b) = \Theta$),

$y -$ количество функциональных символов в системе,

$z -$количество уравнений типа $a = a$ и $\Theta = b$, где Θ не переменная.

Определим отношение $<$ между двумя кортежами, как $\langle x_1\ y_1\ z_1 \rangle < \langle x_2\ y_2\ z_2 \rangle$, если верно одно из следующих условий:

1. $x_1 < x_2$

2. $x_1 = x_2 \ \&\ y_1 < y_2$

3. $x_1 = x_2 \ \&\ y_1 = y_2 \ \&\ z_1 < z_2$

Заметим, что операции (a) и (b) всегда уменьшают z и иногда уменьшают x.
Операция (c) всегда уменьшает y иногда x и, возможно, увеличивает z.
Операция (d) всегда уменьшает x, и иногда увеличивает y.
В случае если у системы нет решений, алгоритм определит это на одном из шагов и завершится.
Иначе с каждой операцией $a - d$ данная тройка будет уменьшаться, а так как $x, y, z \geqslant 0$, данный алгоритм завершится за конечное время. □

Пример.

Исходная система

$$E = \left\{ \begin{array}{c} g\left(x_2\right) = x_1 \\ f\left(x_1,\ h\left(x_1\right),\ x_2\right) = f\left(g\left(x_3\right),\ x_4,\ x_3\right) \end{array} \right\}$$

Применим пункт (c) ко второму уравнению верхней системы получим:

$$E = \left\{ \begin{array}{c} g\left(x_2\right) = x_1 \\ x_1 = g\left(x_3\right) \\ h\left(x_1\right) = x_4 \\ x_2 = x_3 \end{array} \right\}$$

Применим пункт (d) ко второму уравнению верхней системы
(оно изменит 1ое уравнение) получим:

$$E = \left\{ \begin{array}{c} g\left(x_2\right) = g\left(x_3\right) \\ x_1 = g\left(x_3\right) \\ h\left(g\left(x_3\right)\right) = x_4 \\ x_2 = x_3 \end{array} \right\}$$

Применим пункт (c) ко первому ур-ию
и пункт (a) к третьему уравнению верхней системы

$$E = \left\{ \begin{array}{c} x_2 = x_3 \\ x_1 = g\left(x_3\right) \\ x_4 = h\left(g\left(x_3\right)\right) \\ x_2 = x_3 \end{array} \right\}$$

Применим пункт (d) для первого уравнения к последнему уравнению, удалим последнее уравнение и
получим систему в разрешенной форме

$$E = \left\{ \begin{array}{c} x_2 = x_3 \\ x_1 = g\left(x_3\right) \\ x_4 = h\left(g\left(x_3\right)\right) \end{array} \right\}$$

Решение системы:

$$S = \left\{ \begin{array}{c} (x_1 = g(x_3)) \\ (x_2 = x_3) \\ \left(x_4 = h\left(g\left(x_3\right)\right)\right) \end{array} \right\}$$

Утверждение 5.9. Если система имеет решение, алгоритм унификации приводит систему в разрешенную форму

Доказательство. От противного.
Пусть алгоритм завершился и получившаяся система не в разрешенной форме.
Тогда верно одно из следующих утверждений:

1. Одно из уравнений имеет вид отличный от $a_i = \Theta_i$, где a_i – переменная, то есть имеет следующий вид:

   (a) $f_i\ \sigma_1 ... \sigma_n = f_i\ \Theta_1 ... \Theta_n$ – должна быть применена редукция термов $\Rightarrow$ алгоритм не завершился – противоречие.

(b) $f_i\ \sigma_1 ... \sigma_n = a_i$ – должно быть применено правило разворота равенства – противоречие.

2. Все уравнения имеют вид $a_i = \Theta_i$, где a_i – переменная, но a_i встречается в системе больше одного раза.
   В таком случае должно быть применено правило подстановки – противоречие.

$\square$

Определение 5.10. $S \circ T$−композиция подстановок, если $S \circ T = S\ (T\ (a))$

Определение 5.11. S−наиболее общий унификатор, если любое решение (R) системы X может быть получено уточнением: $\exists\, T : R = T \circ S$

Утверждение 5.10. Алгоритм дает наиболее общий унификатор системы, если у нее есть решения.

Доказательство. Пусть S — решение, полученное алгоритмом унификации
R — произвольное решение системы
S_0, R_0 — их сужения на множество переменных соответственно

$$E = \begin{cases} ... \\ a_i = \Theta_i \\ ... \end{cases}$$

где E — разрешенная форма исходной системы
Согласно утверждению 6.9, алгоритм унификации приведет систему в разрешенную форму, и полученное решение S будет иметь сужение S_0, имеющее следующий вид:

1. $S_0(a_l) = \Theta_l$, если a_l входит в левую часть E
2. $S_0(a_r) = a_r$, если a_r входит в правую часть E

Рассмотрим, какой вид может иметь R. Для этого достаточно рассмотреть R_0.
Заметим, что R является решением E, так как E эквивалентна исходной системе.
Следовательно, R_0 имеет следующий вид:

1. $R_0(a_r) = \Theta$, где Θ – произвольный терм, если a_r входит в правую часть E
2. $R_0(a_l) = \Theta_l[a_{r_1} := R_0(a_{r_1}), ..., a_{r_m} := R_0(a_{r_m})]$, где a_{r_k} – переменная из правой части E, если a_l входит в левую часть E

Построим $T : R = T \circ S$. Зададим его через сужение T_0:

1. $T_0(a_r) = R_0(a_r)$, если a_r входит в правую часть E
2. $T_0 = id$, иначе

Покажем, что $R = T \circ S$. Для этого достаточно доказать, что $R_0 = T \circ S_0$
Рассмотрим 2 случая:

1. a_r — переменная из правой части E,
   тогда $(T \circ S_0)(a_r) = T(a_r) = T_0(a_r) = R_0(a_r)$
2. a_l — переменная из левой части E,
   тогда $(T \circ S_0)(a_l) = T(\Theta_l) = \Theta_l[a_{r_1} := R_0(a_{r_1}), ..., a_{r_m} := R_0(a_{r_m})] = R_0(a_l)$

Таким образом, мы для любого решения R предъявили подстановку $T : R = T \circ S$, что является определением того, что S — наиболее общий унификатор. $\square$

6 Лекция 6
Реконструкция типов в просто типизированном λ-исчислении комбинаторы

6.1 Алгоритм вывода типов

Пусть есть: $? \vdash A : ?$, хотим найти пару $\langle$контекст, тип$\rangle$

Алгоритм:

1. Рекурсия по структуре формулы

   Построить по формуле A пару $\langle E, \tau \rangle$, где

   $E-$система уравнений, $\tau-$тип A

2. Решение уравнения, получение подстановки S и из решения E и $S\ (\tau)$ получение ответа

Т.е. необходимо свести вывод типа к алгоритму унификации.

Пункт 6.1. Рассмотрим 3 случая

Обозначение $\rightarrow$ – алгебраический тип

1. $A \equiv x \implies \langle \{\}, \alpha_A \rangle$, где $\{\}-$пустой контекст, α_A-новая переменная, нигде не встречавшаяся до этого в формуле

2. $A \equiv P\,Q \implies \langle E_P \cup E_Q \cup \{\tau_P = \rightarrow\ (\tau_Q\ \alpha_A)\}, \alpha_A \rangle$, где α_A-новая переменная

3. $A \equiv \lambda x.P \implies \langle E_P, \alpha_x \rightarrow \tau_P \rangle$

Пункт 6.2. Алгоритм унификации

Рассмотрим $E-$систему уравнений, запишем все уравнения в алгебраическом виде, т.е. $\alpha \rightarrow \beta \Leftrightarrow \rightarrow \alpha\,\beta$, затем применяем алгоритм унификации.

Лемма 6.1. Рассмотрим терм M и пару $\langle E_M, \tau_M \rangle$, Если $\Gamma \vdash M : \rho$, то существует:

1. $S-$решение E_M тогда $\Gamma = \{x : S(\alpha_x) \mid x \in FV(M)\}$, $FV-$множество свободных переменных в терме M, α_x- переменная, полученная при разборе терма M

   $\rho = S(\tau_M)$

2. Если $S-$ решение E_M, то $\Gamma \vdash M : \rho$,

   Доказательство. индукция по структуре терма M

   (a) Если $M \equiv x$, то так как решение существует, то существует и $S(\alpha_x)$, что: $\Gamma,\ x : S(\alpha_x) \vdash x : S(\alpha_x)$

   (b) Если $M \equiv \lambda x.\,P$, то по индукции уже известен тип P, контекст Γ и тип x, тогда:

   $$\frac{\Gamma,\ x :\ S(\alpha_x) \vdash P :\ S(\alpha_P)}{\Gamma \vdash \lambda x.\,P : S(\alpha_x) \rightarrow S(\alpha_P)}$$

   (c) Если $M \equiv P\,Q$, то по индукции:

   $$\frac{\Gamma \vdash P :\ S(\alpha_P) \equiv \tau_1 \rightarrow \tau_2 \qquad\qquad \Gamma \vdash Q : S(\alpha_Q) \equiv \tau_1}{\Gamma \vdash P\,Q : \tau_2}$$

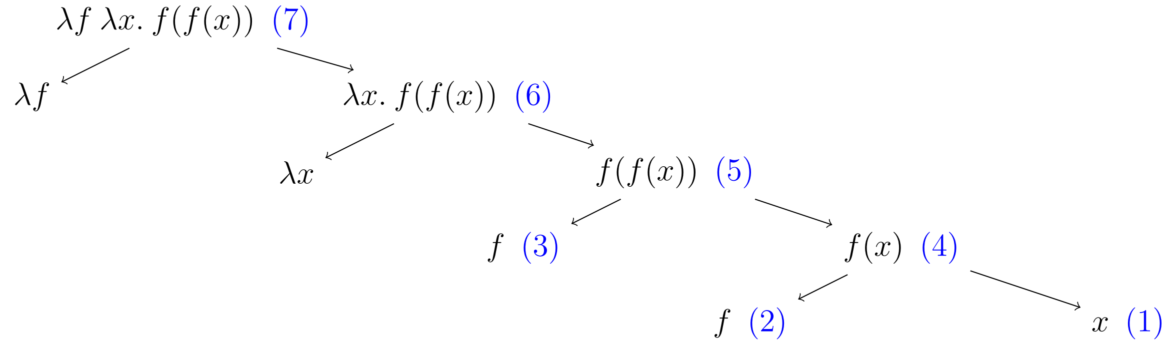
$\square$

$\langle \Gamma, \tau \rangle$ – основная пара для терма M, если

1. $\Gamma \vdash M : \tau$
2. Если $\Gamma' \vdash M : \tau'$, то существует $S:\ S(\Gamma) \subset \Gamma'$

Пример.

Рассмотрим терм: $\lambda f\ \lambda x.\ f(f(x))$, построим и пронумеруем его дерево разбора:



1. $\langle E_1, \tau_1 \rangle = \langle \{\},\ \alpha_x \rangle$
2. $\langle E_2, \tau_2 \rangle = \langle \{\},\ \alpha_f \rangle$
3. $\langle E_3, \tau_3 \rangle = \langle \{\},\ \alpha_f \rangle$
4. $\langle E_4, \tau_4 \rangle = \langle \{\alpha_f = \to (\alpha_x\, \alpha_1)\}, \alpha_1 \rangle$
5. $\langle E_5, \tau_5 \rangle = \left\langle \begin{Bmatrix} \alpha_f = \to (\alpha_x\, \alpha_1) \\ \alpha_f = \to (\alpha_1\, \alpha_2) \end{Bmatrix},\ \alpha_2 \right\rangle$
6. $\langle E_6, \tau_6 \rangle = \left\langle \begin{Bmatrix} \alpha_f = \to (\alpha_x\, \alpha_1) \\ \alpha_f = \to (\alpha_1\, \alpha_2) \end{Bmatrix},\ \alpha_x \to \alpha_2 \right\rangle$
7. $\langle E_7, \tau_7 \rangle = \left\langle \begin{Bmatrix} \alpha_f = \to (\alpha_x \alpha_1) \\ \alpha_f = \to (\alpha_1 \alpha_2) \end{Bmatrix},\ \alpha_f \to (\alpha_x \to \alpha_2) \right\rangle$

$E = \begin{Bmatrix} \alpha_f = \to\ (\alpha_x\, \alpha_1) \\ \alpha_f = \to\ (\alpha_1\, \alpha_2) \end{Bmatrix}$, решим полученную систему:

1. Решим систему:

   (a)
   $$\begin{cases} \alpha_f = \to (\alpha_x\, \alpha_1) \\ \alpha_f = \to (\alpha_1\, \alpha_2) \end{cases}$$

   (b)
   $$\begin{cases} \to (\alpha_1\, \alpha_2) = \to (\alpha_x\, \alpha_1) \end{cases}$$

   (c)
   $$\begin{cases} \alpha_1 = \alpha_x \\ \alpha_2 = \alpha_1 \end{cases}$$

(d)

$$\begin{cases} \alpha_1 = \alpha_x \\ \alpha_2 = \alpha_x \end{cases}$$

2. Получим

$$S = \begin{cases} \alpha_f = \to (\alpha_x\, \alpha_1) \\ \alpha_1 = \alpha_x \\ \alpha_2 = \alpha_x \end{cases}$$

3. $\Gamma = \{\}$, так как в заданной формуле нет свободных переменных

4. Тип терма $\lambda\, f \lambda\, x. f(f(x))$ является результатом подстановки $S(\to\ \alpha_f\ (\alpha_x \to \alpha_2))$, получаем $\tau = (\alpha_x \to \alpha_x) \to (\alpha_x \to \alpha_x)$

6.2 Сильная и слабая нормализации

Определение 6.1. Если существует последовательность редукций, приводящая терм M в нормальную форму, то $M-$слабо нормализуем. (Т.е. при редуцировании терма M мы можем не прийти в н.ф.)

Определение 6.2. Если не существует бесконечной последовательности редукций терма M, то терм $M-$ сильно нормализуем.

Утверждение 6.1.

1. $KI\Omega-$ слабо нормализуема

   Пример.

   Перепишем $KI\Omega$ как $((\lambda x\, \lambda y.\, x)(\lambda x.\, x))(((\lambda x.\, x\, x)(\lambda x.\, x\, x)))$, очевидно, что этот терм можно средуцировать двумя разными способами:

   (a) Сначала редуцируем красную скобку

      i. $((\lambda x\, \lambda y.\, x)(\lambda x.\, x))(((\lambda x.\, x\, x)(\lambda x.\, x\, x)))$

      ii. $((\lambda y.\, (\lambda x.\, x)))(((\lambda x.\, x\, x)(\lambda x.\, x\, x)))$

      iii. $(\lambda x.\, x)$

      Видно, что в этом случае количество шагов конечно.

   (b) Редуцируем синюю скобку. Очевидно, что комбинатор Ω не имеет нормальной формы, тогда понятно, что в этом случае терм $KI\Omega$ никогда не средуцируется в нормальную форму.

2. $\Omega-$ не нормализуема

3. $II-$ сильно нормализуема

Лемма 6.2. Сильная нормализация влечет слабую.

6.3 Выразимость комбинаторов

Утверждение 6.2. Для любого λ-выражения без свободных переменных существует β-эквивалентное ему выражение, записываемое только с помощью комбинаторов S и K, где

$$S = \lambda x\, \lambda y\, \lambda z.\, (x\, z)\, (y\, z) \,:\, (a \to b \to c) \to (a \to b) \to a \to c$$
$$K = \lambda x\, \lambda y.\, x \,:\, a \to b \to a$$

Утверждение 6.3. Комбинаторы S и K являются аксиомами в ИИВ

Утверждение 6.4. Соотношение комбинаторов с λ исчислением:

1. $T(x) = x$
2. $T(P\, Q) = T(P)\, T(Q)$
3. $T(\lambda\, x.P) = K(T(P)),\;\; x \notin FV(P)$
4. $T(\lambda\, x.x) = I$
5. $T(\lambda\, x\lambda\, y.P) = T(\lambda\, x.\, T(\lambda\, y.P))$
6. $T(\lambda\, x.P\, Q) = S\;\, T(\lambda\, x.P)\, T(\lambda\, x.Q)$

Утверждение 6.5. Альтернативный базис:

1. $B = \lambda x\, \lambda y\, \lambda z.\, x\, (y\, z) \,:\, (a \to b) \to (c \to a) \to c \to b$
2. $C = \lambda x\, \lambda y\, \lambda z.\, ((x\, z)\, y) \,:\, (a \to b \to c) \to b \to a \to c$
3. $W = \lambda x\, \lambda y.\, ((x\, y)\, y) \,:\, (a \to a \to b) \to a \to b$

7 Лекция 7

7.1 Импликационный фрагмент ИИП второго порядка

Определение 7.1. Назовем *грамматикой ИИП второго порядка* конструкцию вида:

$$\mathbf{A} ::= (\mathbf{A}) \mid \mathbf{p} \mid \mathbf{A} \to \mathbf{A} \mid \forall \mathbf{p}.\mathbf{A}$$

В этой системе все остальные связки могут быть выражены через основные 4, представленные выше. Например, $\perp$ представима в следующем виде

$$\forall \mathbf{p}.\mathbf{p}$$

Также добавим два новых правила вывода для квантора существования и два для квантора всеобщности к уже существующим в ИИВ:

Для квантора всеобщности:

$$\frac{\Gamma \vdash \phi}{\Gamma \vdash \forall p.\phi}\ (p \notin FV(\Gamma)) \qquad\qquad \frac{\Gamma \vdash \forall p.\phi}{\Gamma \vdash \phi[p := \Theta]}$$

И два для квантора существования:

$$\frac{\Gamma \vdash \phi[p := \psi]}{\Gamma \vdash \exists p.\phi} \qquad\qquad \frac{\Gamma \vdash \exists p.\phi \qquad \Gamma, \phi \vdash \psi}{\Gamma \vdash \psi}\ (p \notin FV(\Gamma, \psi))$$

Определение 7.2. Грамматику ИИП второго порядка с приведенными выше правилами вывода назовем Импликационным фрагментом ИИВ второго порядка

С помощью этих правил вывода можно доказать, что $\bot = \forall p.p$ Действительно, воспользовавшись вторым правилом вывода квантора всеобщности для этого выражения, мы можем вывести любое другое выражение.

С помощью правил вывода также можно доказать, что
$\phi \& \psi \equiv \forall a((\phi \to \psi \to a) \to a)$
$\phi \vee \psi \equiv \forall a((\phi \to a) \to (\psi \to a) \to a)$

Докажем например, что

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash \phi}$$

Воспользуемся вторым правилом вывода для квантора всеобщности

$$\frac{\Gamma \vdash \forall \alpha.\alpha}{\Gamma \vdash \alpha[\alpha := \phi]}$$

7.2 Теория Моделей

Добавим к нашему исчислению модель. Напомню, что модель это функция которая сопоставляет некому терму элемент из множества истинностных значений. В нашем случае мы будем сопоставлять высказываниям элементы из множества $[\![$**И,Л**$]\!]$ по следующим правилам:

$$[\![p]\!] = p,\text{ т. е. } [\![p]\!]^{p=x} = x$$

$$[\![p \to Q]\!] = \begin{cases} \text{Л}, [\![p]\!] = \text{И}, [\![Q]\!] = \text{Л} \\ \text{И}, \text{иначе} \end{cases}$$

$$[\![\forall p.Q]\!] = \begin{cases} \text{И}, [\![Q]\!]^{p=\text{л, и}} = \text{И} \\ \text{Л}, \text{иначе} \end{cases}$$

Эта модель корректна, но не полна.

7.3 Система F

Определение 7.3. Под типом в системе F будем понимать следующее

$$\tau = \begin{cases} \alpha, \beta, \gamma ... & \text{(атомарные типы)} \\ \tau \to \tau \\ \forall \alpha.\tau & (\alpha \text{ - переменная}) \end{cases}$$

Определение 7.4. Введем определение грамматики в системе F:

$$\Lambda ::= \text{x} \mid \lambda x^{\tau}.\Lambda \mid \Lambda\Lambda \mid (\Lambda) \mid \Lambda\alpha.\Lambda \mid \Lambda\tau$$

где $\Lambda\alpha.\Lambda$ — типовая абстракция, явное указание того, что вместо каких-то типов мы можем подставить любые выражения, а $\Lambda\tau$ — это применение типа.

Так, пример типовой абстракции это:

```
template<typename T>
class W {
    T x;
}
```

Типовая аппликация — это объявление переменной класса с каким-то типом

```
W<int> w_test;
```

Теорема 7.1. Изоморфизм Карри - Ховарда:

$$\Gamma \vdash_F M : \tau \Leftrightarrow |\Gamma| \vdash_{\forall,\to} \tau$$

В системе F определены следующие правила вывода:

$$\frac{}{\Gamma, x : \tau \vdash x : \tau} \qquad \frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}$$

$$\frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x^{\tau}.M : \tau \to \sigma} \quad (x \notin FV(\Gamma))$$

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \Lambda\alpha.M : \forall\alpha.\sigma} \quad (\alpha \notin FV(\Gamma)) \qquad \frac{\Gamma \vdash M : \forall\alpha.\sigma}{\Gamma \vdash M\tau : \sigma[\alpha := \tau]}$$

Приведем пример. Покажем как выглядит в системе F левая проекция. В просто типизированном λ - исчислении π_1 имеет тип $\alpha\&\beta \to \alpha$. В системе F явно указывается, что элементы пары могут быть любыми и пишется соответственно $\forall\alpha.\forall\beta.\alpha\&\beta \to \alpha$. Само выражение для проекции также изменится и будет иметь вид $\pi_1 = \Lambda\alpha.\Lambda\beta.\lambda p^{\alpha\&\beta}.p\alpha\mathrm{T}$

Давайте определим еще несколько понятий из простого λ-исчисления. *Начнем с β-редукции:*

1. Типовая β-редукция: $(\Lambda\alpha.M^{\sigma})\tau \to_{\beta} M[\alpha := \tau] : \sigma[\alpha := \tau]$ 2. Классическая β-редукция: $(\lambda x^{\sigma}.M)^{\sigma\to\tau}X \to_{\beta} M[x := X] : \tau$

Выразим еще несколько функций

1. Не бывает M:$\perp$
2. Рассмотрим пару <P, Q> ::= $\Lambda\alpha.\lambda z^{\tau\to\sigma\to\alpha}.zPQ$
Проекторы мы рассмотрели ранее.
3. $in_L(M^{\tau}) ::= \Lambda\alpha.\lambda u^{\tau\to\alpha}.\lambda\omega^{\sigma\to\alpha}.uM$
$in_R(M^{\sigma}) ::= \Lambda\alpha.\lambda u^{\tau\to\alpha}.\lambda\omega^{\sigma\to\alpha}.uM$

(1) Теорема Чёрча-Россера и прочие теоремы, доказуемые в строго-типизированном лямбда-исчислении, доказуемы и в системе F
(2) $\lambda_{(\forall,\to)}$ Система F сильно нормализуема
(3) Y комбинатор не типизируем
(4) Исчисление неразрешимое, но не противоречивое

8 Лекция 8

8.1 Ранг типа

Определение 8.1. Введем определение. Под рангом типа мы будем понимать число, получаемое по следующим правилам:
Rn(x) — множество всех типов x
Rn(0) — все типы без кванторов
Rn(x+1) = Rn(x) | Rn(x) → Rn(x+1) | $\forall \alpha . Rn(x+1)$

Примеры 1. α ∈Rn(0)
2. $\forall \alpha . \alpha$ ∈Rn(1)
3. $(\forall \alpha . \alpha) \to (\forall \beta . \beta)$ ∈Rn(2), так как каждый тип вида $\forall \alpha . \alpha$ ∈Rn(1), то по третьему правилу весь тип ∈Rn(2)

Определение 8.2. Тип с поверхностными кванторами — это любой тип вида $\forall \alpha . \tau$, где в τ отсутствуют кванторы. Очевидно, что любой такой тип ∈Rn(1). Действительно, тип внутри квантора точно имеет ранг 0. Навешивание одного или нескольких кванторов всеобщности увеличит его ранг на единицу.

8.2 Типовая схема

Возьмем только типы с поверхностными кванторами(из Rn(1)).
Также можно превратить любую формулу из Rn(1) в формулу с поверхностными кванторами.
Например:
$\beta \to \forall \alpha . \alpha \equiv \forall \alpha . (\beta \to \alpha)$

Определение 8.3. Типовой схемой назовем выражение вида:

$$\sigma \equiv \forall \alpha_1 . \forall \alpha_2 ..... \forall \alpha_n . t \text{ где t} \in \text{Rn}(0)$$

Также будем считать, что $\sigma_1 <= \sigma_2$
(σ_1 является спецификацией σ_2) если:

$\sigma_2 \equiv \forall \alpha_1 ... \forall \alpha . \tau_1$
$\sigma_1 \equiv \forall \beta_1 ... \beta_n . \tau_1 [\alpha_1 := \Theta_1] ... [\alpha_n := \Theta_n]$
Например:
$\forall \beta_1 . \forall \beta_2 . (\beta_1 \to \beta_2) \to (\beta_1 \to \beta_2)$
является спецификацией $\forall \alpha . \alpha \to \alpha$

8.3 Экзистенциальные типы

1) $\dfrac{\Gamma \vdash \phi[\alpha := \theta]}{\Gamma \vdash \exists \alpha . \phi}$

2) $\dfrac{\Gamma \vdash \exists \alpha . \phi \qquad \Gamma, \phi \vdash \psi}{\Gamma \vdash \psi}$

Экзистенциальные типы это способ инкапсуляции данных. Предположим, что у нас есть стек с хранилищем типа α, у которого определены следующие операции:

empty: α
push: $\alpha \& \nu \to \alpha$
pop: $\alpha \to \alpha \& \nu$

Тогда очевидно, что тип stack$\equiv \alpha \& (\alpha \& \nu \to \alpha) \& (\alpha \to \alpha \& \nu)$. Но что если мы реализовали хранилище как-то по-особенному, не меняя типов операций. Мы хотим скрыть данные о реализации, в частности о типе α. Вместо деталей просто скажем, что существует интерфейс, удовлетворяющий такому типу: $\exists \alpha . \alpha \& (\alpha \& \nu \to \alpha) \& (\alpha \to \alpha \& \nu)$

8.4 Абстрактные типы

Предположим, что мы захотим создать стек, в котором лежат целые числа. Рассмотрим, как тогда будет выглядеть тип созданного стека:
stack$\equiv \forall \nu . \exists \alpha . \alpha \& (\alpha \& \nu \to \alpha) \& (\alpha \to \alpha \& \nu)$
По аналогии с правилом удаления квантора существования, можно определить правила вывода для выражений абстрактных типов:

$$\frac{\Gamma \vdash M : \varphi[\alpha := \theta]}{\Gamma \vdash (\text{pack } M, \theta \text{ to } \exists \alpha . \varphi) : \exists \alpha . \varphi}$$

Это правило вывода позволяет скрыть реализацию стека, так как если α — это тип стека, то $\alpha[\nu := \theta]$ — его конкретная реализация, например ArrayStack, LinkedListStack и подобные

$$\frac{\Gamma \vdash M : \exists \alpha . \varphi \qquad \Gamma, x : \varphi \vdash N : \psi}{\Gamma \vdash \text{abstype } \alpha \text{ with } x : \varphi \text{ in } M \text{ is } N : \psi} (\alpha \notin FV(\Gamma, \psi))$$

Это правило вывода соответствует виртуальному вызову стека какой-то реализации, например:

```
foo(Stack s) {
...
```

}

Поскольку выводимые формулы выглядят слишком громоздко, перепишем их, вспомнив, что:
$\exists\alpha.\sigma \equiv \forall\beta.(\forall\alpha.\sigma \to \beta) \to \beta$

Тогда:
pack M, θ **to** $\exists\alpha.\varphi = \Lambda\beta.\lambda x^{\forall\alpha.\varphi\to\beta}.x\theta M$
abstype α **with** $x : \varphi$ **in** M **is** $N : \psi = M\psi(\Lambda\alpha.\lambda x^{\varphi}.N)$

8.5 Типовая система Хиндли-Милнера

Начнем с определения типа. Тип в системе Хиндли-Милнера:

Монотип — выражение в грамматике вида $\tau ::= \alpha|\tau \to \tau|(\tau)$
Политип — выражение в грамматике вида $\sigma ::= \tau|\forall\alpha.\sigma$

Поэтому типы вида $\alpha \to \forall\beta.\beta$ - некорректны в системе ХМ

Грамматика в системе Хиндли-Милнера имеет вид:

$$\Lambda ::= x|\lambda x.\Lambda|\Lambda\Lambda|(\Lambda)|\text{let x} = \Lambda \text{ in } \Lambda$$

Хотя в системе Хиндли-Милнера (как и во всех рассматриваемых нами типовых системах) нельзя типизировать $\mathcal{Y}$-комбинатор, можно добавить его, расширив язык. Давайте определим его как $\mathcal{Y}f = f\,(\mathcal{Y}f)$. Какой у него должен быть тип? Пусть $\mathcal{Y}$ принимает f типа α, и возвращает нечто типа β, то есть $\mathcal{Y} : \alpha \to \beta$. Функция f должна принимать то же, что возвращает $\mathcal{Y}$, так как результат $\mathcal{Y}$ передаётся в f, и возвращать она должна то же, что возвращает $\mathcal{Y}$, так как тип выражений с обеих сторон равенства должен быть одинаковый, то есть $f : \beta \to \beta$ Кроме того, α и тип f это одно и то же, $\alpha = \beta \to \beta$. После подстановки и заключения свободной переменной под квантор получаем $\mathcal{Y} : \forall\beta.(\beta \to \beta) \to \beta$.

Через такой $\mathcal{Y}$ можно определять рекурсивные функции, и они будут типизироваться.

Правила вывода для системы Хиндли-Милнера будут даны в следующей лекции.

9 Лекция 9

Определение 9.1 (Ранг типа). $R(x)$ — все типы ранга x.

- $R(0)$ — все типы без кванторов
- $R(x+1) = R(x) \mid R(x) \to R(x+1) \mid \forall\alpha.R(x+1)$

Например:

- $\alpha \in R(0)$
- $\forall\alpha.\alpha \in R(1)$
- $(\forall\alpha.\alpha) \to (\forall b.b) \in R(2)$
- $((\forall\alpha.\alpha) \to (\forall b.b)) \to b \in R(3)$

Тут видно, если выражение слева от знака импликации имеет ранг n, то все выражение будет иметь ранг $\geqslant (n+1)$.

Утверждение: Пусть x — выражение только с поверхностными кванторами, тогда $x \in R(1)$.

Определение 9.2 (Типовая схема)**.**
$\sigma ::= \forall\alpha_1.\forall\alpha_2.\dots\forall\alpha_n.\tau$, где $\tau \in R(0)$ и, следовательно, $\sigma \in R(1)$.

Определение 9.3 (Частный случай (специализация) типовой схемы)**.**
σ_1, σ_2 — типовые схемы
σ_2 — частный случай σ_1 (обознается как $\sigma_1 \sqsubseteq \sigma_2$), если

1. $\sigma_1 = \forall\alpha_1.\forall\alpha_2.\dots\forall\alpha_n.\tau_1$
2. $\sigma_2 = \forall\beta_1.\forall\beta_2.\dots\forall\beta_m.\tau_1[\alpha_i := S(\alpha_i)]$
3. $\forall i.\beta_i \in FV(\tau_1)$

Пример.
$\forall\alpha.\alpha \to \alpha \sqsubseteq \forall\beta_1.\forall\beta_2 : (\beta_1 \to \beta_2) \to (\beta_1 \to \beta_2)$
Вполне возможно, что в ходе замены, все типы будут уточнены (α уточнится как $\beta_1 \to \beta_2$).

9.1 Хиндли-Милнер

1. Все типы только с поверхностными кванторами ($R(1)$)
2. $\overline{HM} ::= p \mid \overline{HM}\ \overline{HM} \mid \lambda p.\overline{HM} \mid let = \overline{HM}\ in\ \overline{HM}$

- $\exists p.\phi = \forall b.(\forall p.(\phi \to b)) \to b$
- $\phi \to \perp \equiv \forall b.(\phi \to b)$
- $$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, \forall p.(\phi \to b) \vdash \forall p.(\phi \to b)}{\Gamma, \forall p.(\phi \to b) \vdash \phi[p := \Theta] \to b}}{\Gamma, \forall p.(\phi \to b) \vdash b}}{\Gamma \vdash (\forall p.(\phi \to b)) \to b}}{\Gamma \vdash \forall b.(\forall p.(\phi \to b)) \to b}$$

Соглашение:

- σ — типовая схема
- τ — простой тип

1. $\overline{\Gamma, x : \sigma \vdash x : \sigma}\ x \notin \mathrm{FV}(\Gamma)$
2. $\dfrac{\Gamma \vdash e_0 : \tau \to \tau' \qquad \Gamma \vdash e_1 : \tau}{\Gamma \vdash e_0\ e_1 : \tau'}$
3. $\dfrac{\Gamma, x : \tau \vdash e : \tau'}{\Gamma \vdash \lambda x.e : \tau \to \tau'}$
4. $\dfrac{\Gamma \vdash e_0 : \sigma \qquad \Gamma, x : \sigma \vdash e_1 : \tau}{\Gamma \vdash let\ x = e_0\ in\ e_1 : \tau}$, $let\ x = a\ in\ b \equiv (\lambda x.b)\ a$
5. $\dfrac{\Gamma \vdash e : \sigma' \qquad \sigma' \sqsubseteq \sigma}{\Gamma \vdash e : \sigma}$
6. $\dfrac{\Gamma \vdash e : \sigma}{\Gamma \vdash e : \forall \alpha.\sigma}\ \alpha \notin FV(\Gamma)$

9.2 Алгоритм вывода типов в системе Хиндли-Милнера W

На вход подаются $\Gamma,\ M$, на выходе наиболее общая пара (S, τ)

1. $M = x,\ x : \tau \in \Gamma$ (иначе ошибка)
   - Выбросить все кванторы из τ
   - Переименовать все свободные переменные в свежие
     Например: $\forall \alpha_1.\phi \Rightarrow \phi[\alpha_1 := \beta_1]$, где β_1 — свежая переменная

   $(\varnothing, \Gamma(x))$
2. $M = \lambda n.e$
   - τ — новая типовая переменная
   - $\Gamma' = \Gamma \backslash \{n : \_\}$ (т.е. Γ без переменной n)
   - $\Gamma'' = \Gamma' \cup n : \tau$
   - $(S',\ \tau') = W(\Gamma'', e)$

   $(S', S'(\tau) \to \tau')$
3. $M = P\ Q$
   - $(S_1, \tau_1) = W(\Gamma, P)$
   - $(S_2, \tau_2) = W(S_1(\Gamma), Q)$
   - S_3 — Унификация $(S_2(\tau_1), \tau_2 \to \tau)$

$(S_3 \circ S_2 \circ S_1, S_3(\tau))$

4. $let\ x = P\ in\ Q$

   - $(S_1, \tau_1) = W(\Gamma, P)$
   - $\Gamma' = \Gamma$ без x
   - $\Gamma'' = \Gamma' \cup \{x : \forall \alpha_1 \dots \alpha_k . \tau_1\}$, где $\alpha_1 \dots \alpha_k$ — все свободные переменные в τ_1
   - $(S_2, \tau_2) = W(S_1(\Gamma''), Q)$

   $((S_2 \circ S_1), \tau_2)$

Надеемся, что логика второго порядка противоречива.

9.3 Рекурсивные типы

Ранее мы уже рассматривали Y-комбинатор, но не могли типизировать его и отказывались. Однако в программировании хотелось бы использовать рекурсию, поэтому тут мы введем его аксиоматически.

$Y f =_\beta f(Y\ f)$

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

И теперь, когда мы хотим написать какую-то рекурсивную функцию, скажем, на языке Ocaml, то интерпретировать ее можно будет следующим образом:

```
let rec f = expr in
    expression
```

```
let f = Y (λ f. expr) in
    expression
```

Рекурсивными могут быть не только функции, но и типы. Как, например, список из целых чисел:

```
type intList = Nil | Cons of int * intList;;
```

На нем мы можем вызывать рекурсивные функции, например, ниже представлен фрагмент кода, позволяющий найти длину списка.

```
let rec length l = match l with
  | Nil -> 0
  | Cons (x, s) -> 1 + length s;;

let my_list = Cons(1, Cons (2, Cons (3, Nil)));;

print_int (length my_list);;  (* output: 3 *)
```

Рассмотрим, что из себя представляет тип списка выше:
$Nil = inLeft\ O = \lambda a.\lambda b.a\ O$
$Cons = inRight\ p = \lambda a.\lambda b.b\ p$
$\lambda a.\lambda b.a\ O : \forall\gamma.(\alpha \to \gamma) \to (\beta \to \gamma) \to \gamma$
$\lambda a.\lambda b.b\ p : \forall\gamma.(\alpha \to \gamma) \to (\beta \to \gamma) \to \gamma$
$\delta = \forall\gamma.(\alpha \to \gamma) \to (\beta \to \gamma) \to \gamma$
$\lambda a.\lambda b.b\ (\lambda a.\lambda b.a\ O) : \forall\alpha.(\alpha \to \gamma) \to (\delta \to \gamma) \to \gamma$

Научимся задавать рекурсивные типы, а именно рассмотрим два способа решения:

1. Эквирекурсивный

   ```
   list = Nil | Cons a * list
   ```

   $\alpha = f(\alpha)$ — уравнение с неподвижной точкой. Пусть $\mu\alpha.f(\alpha) = f(\mu\alpha.f(\alpha))$. Используем это в типах, μf — это и есть тип списка. То есть мы ввели оператор μ, действующий на типах, аналогично Y-комбинатору для выражений.

   На практике такой подход используется и в языке программирования Java:

   ```
   class Enum <extends Enum<E>>
   ```

   Также приведем пример вывода типа $\lambda x.x\ x$ (можно вспомнить, что именно этот терм помешал нам типизировать Y-комбинатор в просто типизированном λ-исчислении):

   Пусть $\tau = \mu\alpha.\alpha \to \beta$. Если мы раскроем τ один раз, то получим $\tau = \tau \to \beta$. Если раскроем еще раз, то получим $\tau = (\tau \to \beta) \to \beta$.

   $$\dfrac{\dfrac{x : \tau \vdash x : \tau \to \beta \qquad x : \tau \vdash x : \tau}{x : \tau \vdash x\ x : \beta}}{\vdash \lambda x.x\ x : \tau \to \beta}$$

   Ранее мы ввели Y-комбинатор аксиоматически, а можем ли мы его типизировать используя рекурсивные типы? Ответ: Да, можем. Напомним, что $Y = \lambda f.(\lambda x.f\ (x\ x))\ (\lambda x.f\ (x\ x))$.

$$
\dfrac{
\dfrac{\dfrac{\dfrac{\lambda f:\beta \to \beta,\ x:\tau \vdash f:\beta \to \beta \quad f:\beta \to \beta,\ x:\tau \vdash x\ x:\beta}{f:\beta \to \beta, x:\tau \vdash f\ (x\ x)}}{f:\beta \to \beta \vdash \lambda x.f\ (x\ x):\tau}}{\lambda f:\beta \to \beta \vdash \lambda x.f\ (x\ x):\tau \to \beta}
\qquad
\dfrac{\text{аналогично}}{\lambda f:\beta \to \beta \vdash \lambda x.f\ (x\ x):\tau}
}{
\dfrac{\dfrac{f:\beta \to \beta \vdash (\lambda x.f\ (x\ x))\ (\lambda x.f\ (x\ x)):\beta}{\vdash \lambda f.(\lambda x.f\ (x\ x))\ (\lambda x.f\ (x\ x)):(\beta \to \beta) \to \beta}}{\vdash \lambda f.(\lambda x.f\ (x\ x))\ (\lambda x.f\ (x\ x)):\forall \beta.(\beta \to \beta) \to \beta}
}
$$

Загадочка: А можно ли типизировать, скажем $\lambda x : Nat.x(Sx)$?

2. Изорекурсивный

   В отличие от эквирекурсивных типов будем считать, что $\mu\alpha.f(\alpha)$ изоморфно $f(\mu\alpha.f(\alpha))$. Такой подход используется в языке программирования C.

   ```
   struct list {
       list* x;
       int a;
   }
   (*x).(*x).(*x).a
   // или, что эквивалентно
   x->x->x.a
   ```

   Можно заметить, что выше для работы со списком мы использовали специальную операцию: $* : list* \to list$ — разыменование

   В изорекурсивных типах введены специальные операции для работы с этими типами, и оператор * из C как раз был примером одной из них (в частности roll):

   - $Roll : Nil|Cons(a * list) \to list$
   - $Unroll : list \to Nil|Cons(a * list)$

   В более общем виде (введение в типовую систему):

   - $roll : f(\alpha) \to \alpha$
   - $unroll : \alpha \to f(\alpha)$

   Можно привести еще примеры из языка C:

   - $* : T* \to T$
   - $\& : T \to T*$
   - $T = \alpha$
   - $T* = f(\alpha)$

9.4 Зависимые типы

Рассмотрим функцию sprintf из языка C:

$sprintf : string \to smth \to string$

$sprintf "\%d" : int \to string$

$sprintf "\%f" : float \to string$

Легко видеть, что тип sprintf определяется первым аргументом. То есть тип этой функции зависит от терма — именно такой тип и называется зависимым (*англ: dependent type*).

Рассмотрим несколько иной пример, а именно список. Предположим, что мы хотим скалярно перемножить два списка:

```
let rec dot lst1 lst2 = match (lst1, lst2) with
  | ([], []) -> 0
  | (x :: xs, y :: ys) -> x * y + (dot xs ys)
;;

dot [1; 2] [3; 4] (* results in 11 *)

dot [1; 2] [3; 4; 5] (* получим ошибку *)
```

Было бы очень здорово уметь отлавливать эту ошибку не в рантайме, а во время компиляции программы и зависимые типы могут в этом помочь. Например в языке Idris можно использовать `Vect`:

```
dot : {n : Nat} -> Vect n Integer -> Vect n Integer -> Integer
dot {n = Z} [] [] = 0
dot {n = (S len)} (x :: xs) (y :: ys) = y * x + dot xs ys

let v1 = Data.Vect.fromList [1, 2, 3]
let v2 = Data.Vect.fromList [4, 5, 6]
dot v1 v2 -- results in 32

let v1 = Data.Vect.fromList [1, 2, 3, 4]
dot v1 v2 -- Type mismatch between
          --      Vect 3 Integer (Type of v2)
          -- and
          --      Vect 4 Integer (Expected type)
```

Если подойти к типу функции `dot` ближе с точки зрения теории типов, то мы бы записали это так (о * речь пойдет в следующей главе [стоит ее воспринимать как тип типа]):

`Nat:*, Integer:*, Vect : Nat -> Integer -> *` $\vdash$
Π `n:Nat . Vect n Integer -> Vect n Integer -> Integer`

9.4.1 П-типы и Σ-типы

- $\Pi x : \alpha.P(x)$ - эту запись можно читать как (в каком-то смысле в интуиционистском понимании): "У меня есть метод для конструирования объекта типа $P(x)$, использующий любой предоставленный x типа α". Если же смотреть на эту запись с точки зрения классической логики, то ее можно понимать как бесконечную конъюнкцию $P(x_1)\&P(x_2)\&...$. Данная конъюнкция соответствует декартовому произведению, отсюда и название П-типа (иногда в англоязычной литературе можно встретить *dependent function type*).

- $\Sigma x : \alpha.P(x)$. Аналогично предыдущему пункту рассмотрим значение с интуиционистской точки зрения: "У меня есть объект x типа α, но больше ничего про него не знаю кроме того, что он обладает свойством $P(x)$". Это как раз в стиле интуиционизма, что нам приходится знать и объект x и его свойство $P(x)$. Это можно представить как пару, а пара - бинарное произведение. С точки же зрения классической логики, мы можем принимать эту формулу как бесконечную дизъюнкцию $P(x_1) \vee P(x_2) \vee ...$, которая соответствует алгебраическим типам данных. (иногда в англоязычной литературе можно встретить *dependent sum*).

Ранее обсуждалось, что тип может быть сопоставлен множеству его значений, как например тип uint32_t в C++ может быть сопоставлен множеству $\{0, 1, ..., 2^{32} - 1\}$. Рассмотрим $\Pi x : \alpha.P(x)$: этому П-типу можно сопоставить прямое произведение B^A (где A — множество, сопоставленное типу α, а $B(a)$ — множество, сопоставленное типу $P(a)$), которое следует воспринимать, как $B^A = \prod_{a\in A} B(a) = \{f : A \to \bigcup_{a\in A} B(a) \mid f(a) \in B(a), a \in A\}$. Можно отметить, что если $B(a) = C = const$, то на любой вход $f(a) \in C$, т.е. тип значения $f(a)$ не меняется, собственно поэтому этот тип в таком случае записывают как $A \to P$. Рассмотрим $\Sigma x : \alpha.P(x)$: этому Σ-типу можно сопоставить дизъюнктное объединение $\sqcup_{a\in A} B(a) = \bigcup_{a\in A}\{(a, x) | x \in B(a)\}$, где A — множество, сопоставленное типу α, а $B(a)$ - множество, сопоставленное типу $P(a)$. Тут также можно отметить, что если $B(a) = C = const$, то результатом дизъюнктивного объединения будет прямое произведение $A \times B$. В языке программирования Idris примером Σ-типа является зависимая пара:

```
data DPair : (a : Type) -> (P : a -> Type) -> Type where
  MkDPair : {P : a -> Type} -> (x : a) -> P x -> DPair a P
```

Также есть некоторый синтаксический сахар (a : A ** P), который обозначает зависимую пару типа DPair A P, где P может содержать в себе имя a.

В документации Idris'а есть хороший пример использования: мы хотим отфильтровать вектор (Vect) по какому-то предикату - мы не можем знать заранее длину результирующего вектора, поэтому зависимая пара выручает:

```
filter : (a -> Bool) -> Vect n a -> (p ** Vect p a)
filter p Nil = (_ ** [])
filter p (x :: xs) with (filter p xs)
  | ( _ ** _xs ) = if (p x) then
                     ( _ ** x :: _xs )
                   else
                     ( _ ** _xs )
```

10 Лекция 10

10.1 Введение

Прежде мы разбирали просто типизированное лямбда-исчисление, в котором термы зависели от термов, например, терм $(F\ M)$ зависит от терма M. После того, как было замечено, что, скажем, I может иметь разные типы, которые по сути различаются лишь аннотацией, например, $\lambda x.x : \alpha \to \alpha$, $\lambda x.x : (\alpha \to \alpha) \to (\alpha \to \alpha)$, была введена типовая абстракция, то есть термы теперь могли зависеть от типов и такая типовая система была названа System F и можно было писать $\Lambda\alpha.\lambda x : \alpha.x : \forall\alpha.\alpha \to \alpha$. То есть это было своего рода изобретением шаблонов в языке C++. Но на этом все не ограничено. System F_w, в которой типы могут зависеть от типов, как, например, список - алгебраический тип данных, у которого есть две альтернативы $Nil : \forall\alpha.List\alpha$ и $Cons : \forall\alpha.\alpha \to List\alpha \to \alpha$ (рекурсивные типы смотри выше). Для лучшего понимания различия системы F и F_w ниже представлены грамматики для типов:

- $T_\to ::= \alpha \mid (T_\to) \mid T_\to \to T_\to$
- $T_F ::= \alpha \mid \forall\alpha.T_F \mid (T_F) \mid T_F \to T_F$
- $T_{F_w} ::= \alpha \mid \lambda\alpha.T_{F_w} \mid (T_{F_w}) \mid T_{F_w} \to T_{F_w} \mid T_{F_w}\ T_{F_w}$

Ничего не мешает рассматривать типовую систему, в которой тип может зависеть от терма, как это было сделано раньше. Пусть для всех $a : \alpha$ мы можем определить тип β_α и пусть существует $b_\alpha : \beta_\alpha$. Тогда вполне обоснована запись функции $\lambda\alpha : b_\alpha$. Тип данного выражения принято записывать как $\Pi a : \alpha.\beta_\alpha$ (стоит сделать замечание, что если β_α не зависит от α [то есть функция константа], то вместо $\Pi a : \alpha.\beta_\alpha$ пишут $\alpha \to \beta$). Примером может быть тип вектора, длина которого зависит от натурального числа и типа (пример из языка Idris):

```
data Vect : (len : Nat) -> (elem : Type) -> Type where
  Nil   : Vect Z elem
 (::) : (x : elem) -> (xs : Vect len elem) -> Vect (S len) elem
```

Теперь наша грамматика стала обширной и появилась необходимость более формально говорить о типах, т.е. ввести их в систему. Для этого был придуман род (*англ: kind*), который обозначают $*$. Используя $*$ можно задавать типы типовых конструкторов.

Рассмотрим пару примеров, как используется род:

- $\lambda m : \alpha.F\ m : (\alpha \to \beta) : *$
- $\lambda \alpha : *.I_\alpha : (\Pi \alpha : *.\alpha \to \alpha) : *$
- $\lambda n : Nat.A^n \to B : Nat \to *$
- $\lambda a : *.a \to a : * \to *$

Попробуем разобраться, что же написано в примерах.

- Первый пример — это типизация привычной нам абстракции. Утверждение $a \to b : *$ значит $a \to b$ — это тип.
- Во втором примере мы рассматриваем лямбда-выражение, которое принимает на вход тип и возвращает терм I_α. Таким образом мы собираемся типизировать терм, зависящий от типа. Для этого как сказано выше мы вводим символ Π, а вот в известной нам системе F тип выражения $\lambda \alpha : *.I_\alpha$ был бы $\forall \alpha.(\alpha \to \alpha)$.
- В третьем пункте мы хотим сформировать утверждения для типа, зависящего от терма. Интуитивно понятно, что у такого выражения будет род $Nat \to *$. И заселять его будут конструкторы типов, которые принимают на вход число и возвращают тип, например $\lambda x : Nat.int[x]$ — это терм, который заселяет род $Nat \to *$
- В четвертом пункте мы типизируем конструктор типа, который принимает на вход тип. Действительно, его родом будет $* \to *$.

Возникает желание каким-то образом объединить все роды, и это необходимо для дальнейшей формализации происходящего. $* \to * :?$. Что можно поставить на место вопросика? Это не тип, так как иначе бы могли записать $* \to * : *$, однако понятно, что это не так. В частности, для этого вводится понятие сорта (*англ. sort*), которое можно воспринимать как тип рода и тогда $* \to * : \square$ и $* : \square$. Для любого выражения вида $A \to *$, где A — это что угодно, верно, что оно типизируется $\square$. Например,

$* \to * \to * : \Box$ - этот род очень похож на $* \to *$, и действительно, единственное отличие заключается в количестве аргументов нашего типового конструктора. В частности, этот род заселяет конструктор map, $\lambda keyType : *.(\lambda valueType.map < keyType, valueType >)$

Теперь мы ознакомились со всеми необходимыми обозначениями и неформальными определениями. Обобщая все вышесказанное, построим обобщенную типовую систему.

10.2 Обобщенная типовая система

- Сорта: {*, $\Box$}
  - Выражение "$A : *$"означает, что A — тип. И тогда, если на метаязыке мы хотим сказать "Если A тип, то и $A \to A$ тоже тип то формально это выглядит как $A : * \vdash (A \to A) : *$
  - $\Box$ - это абстракция над сортом для типов.
  - Например:
    - $5 : int : * : \Box$
    - $[] : * \to * : \Box$
    - $\Lambda M.List < M >: * \to * : \Box$
- $T ::= x \mid c \mid T\ T \mid \lambda x : T.\ T \mid \Pi x : T.\ T$
- Аксиома:
  - $\dfrac{}{\vdash * : \Box}$
- Правила вывода:
  1. $\dfrac{\Gamma \vdash A : S}{\Gamma, x : A \vdash x : A}\ x \notin \Gamma$
  2. $\dfrac{\Gamma \vdash A : B \qquad \Gamma \vdash C : S}{\Gamma, x : C \vdash A : B}$ — правило ослабления (примерно как $\alpha \to \beta \to \alpha$ в
  3. $\dfrac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : S \qquad B =_\beta B'}{\Gamma \vdash A : B'}$ — правило конверсии
  4. $\dfrac{\Gamma \vdash F : (\Pi x : A.B) \qquad \Gamma \vdash a : A}{\Gamma \vdash (F\ a) : B[x := a]}$ — правило применения
- Семейства правила (generic-правила)

  Пусть $(s_1, s_2) \in S \subseteq \{*, \Box\}^2$.

  1. П-правило: $\dfrac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\Pi x : A.B) : s_2}$

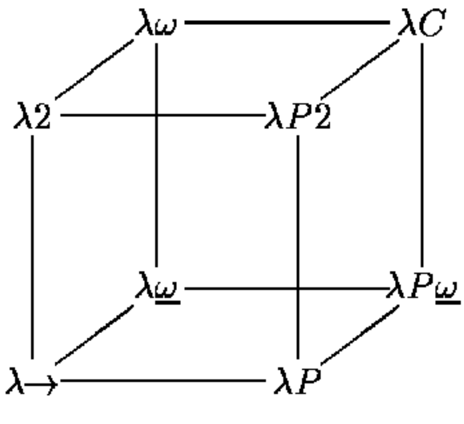
2. λ-правило: $\dfrac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash b : B \qquad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\lambda x : A.b) : (\Pi x : A.B)}$

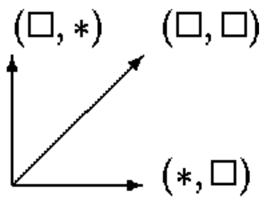
В одном из примеров мы рассмотрели утверждение $\lambda\alpha : *.I_\alpha : (\Pi\alpha : *.\alpha \to \alpha) : *$. Теперь мы можем до конца понять, почему $(\Pi\alpha : *.\alpha \to \alpha) : *$ и что такое П. Неформально говоря, П-правило говорит нам о том, что выражение $(\Pi x : A.B)$ типизируется либо $*$, либо $\Box$, а именно тем, чем является B. То есть, $(\Pi x : A.B)$ — это либо тип конструктора типа, либо тип конструктора терма. В приведенном примере мы принимаем на вход любой тип α и возвращаем терм, а значит $(\Pi\alpha : *.\alpha \to \alpha) : *$.

Еще пару слов про П. Этот символ является обобщением $\to$, поэтому, во всех рассмотренных ранее родах, согласно нашей обобщенной типовой системе, можно заменить $\to$ на П, согласно замечанию выше. Например, $* \to * = \Pi a : *.*$. Важно понимать, что подразумевается под зависимостью тела от аргумента и не путать понятия терм и тип. В $\Pi a : *.*$ тело не зависит от аргумента, потому что тело — это просто звездочка, то есть $\Pi a : *.*$ говорит нам просто о том, что наше выражение принимает тип и выдает тип. В то время как термы, населяющие $\Pi a : *.*$, разумеется, могут иметь тело, зависящее от аргумента, как, например, $\lambda a : *.a \to a$

10.3 λ-куб

В обобщенных типовых системах есть generic-правила, которые зависят от выбора s_1 и s_2 из множества сортов. Этот выбор можно проиллюстрировать в виде куба.





Выбор правил означает следующее:

- $(*,\ *)$ - позволяет записывать термы, которые зависят от термов
- $(\Box,\ *)$ - позволяет записывать термы, которые зависят от типов
- $(*,\ \Box)$ - позволяет записывать типы, которые зависят от термов
- $(\Box,\ \Box)$ - позволяет записывать типы, которые зависят от типов

На самом деле в данной формулировке под типом понимается не только привычный тип. Потому что для привычного типа верно $\tau : *$. Здесь же τ может типизироваться чем угодно, кроме $\Box$. В частности $* \to *$, это значит, что например std::vector<T> тоже подходит.

Также на этом кубике можно расположить языки программирования, например:

- Haskell будет располагаться на левой грани куба, недалеко от λw
- Idris и Coq, очевидно, будет находиться в λC
- C++ очень ограниченно приближается к λC (мысли вслух):
  1. $(*,\ *)$ - без этого не может обойтись ни один язык программирования
  2. $(\Box,\ *)$ - например, sizeof(type)
  3. $(*,\ \Box)$ - например, std::array<int, 19> - тут есть ограничение на то, значение каких типов можно подставлять.
  4. $(\Box,\ \Box)$ - например, std::vector<int>, int*

10.4 Свойства

Для систем в λ-кубе верны следующие утверждения:

- **Th. SN** Обобщенная типовая система сильно нормализуема
- **Th. Черча-Россера**
  1. Для любых трёх элементов A, B и C, таких, $A \twoheadrightarrow B$ и $A \twoheadrightarrow C$ верно, что существует D, что $B \twoheadrightarrow D$ и $C \twoheadrightarrow D$
  2. Для любых двух элементов A, B, для которых верно $A =_\beta B$, существует C, что $A \twoheadrightarrow C$ и $B \twoheadrightarrow C$
- **Th. Subject reduction** $\Gamma \vdash A : T$ и $A \twoheadrightarrow B$, тогда $\Gamma \vdash B : T$
- **Th. Unicity of types** $\Gamma \vdash A : T$ и $\Gamma \vdash A : T'$ тогда $T =_\beta T'$

Примеры:

- $\lambda\omega$:

$$\vdash (\lambda\alpha : *.\alpha \to \alpha) : (* \to *) : \Box$$

1. $$\dfrac{\vdash * : \Box \qquad \dfrac{\vdash *.\Box}{a : * \vdash *.\Box}}{\vdash (* \to *) : \Box}$$

2. $$\dfrac{\vdash * : \Box \qquad \dfrac{\alpha : * \vdash \alpha : * \qquad \alpha : *, x : \alpha \vdash \alpha : *}{\alpha : * \vdash \alpha \to \alpha : x} \qquad \dfrac{\vdash * : \Box}{a : * \vdash * : \Box}}{\vdash (\lambda\alpha : *.\alpha \to \alpha) : * \to *}$$

Notes:

- $(\lambda x.x) : (A \to A)$ - implicit typing (Curry style)
- $I_A = \lambda x : A.x$ - explicit typing (Church style)

Рассмотрим еще примеры для улучшения понимания лямбда-куба и обобщенной типовой системы:

- В системе F ($\lambda 2$) выводимо:
  1. $\vdash (\lambda\alpha : *.\lambda a : \alpha.a) : (\Pi\alpha : *.(\alpha \to \alpha)) : *$
  2. $A : * \vdash (\lambda\alpha : *.\lambda a : \alpha.a)A : (A \to A)$
  3. $A : *, b : A \vdash (\lambda\alpha : *.\lambda a : \alpha.a)Ab : A$

     Разумеется, здесь имеет место редукция: $(\lambda\alpha : *.\lambda a : \alpha.a)Ab \to_\beta b$.
- В $\lambda\underline{w}$ выполняется
  1. $\vdash (\lambda\alpha : *.\alpha \to \alpha) : * \to * : \Box$
  2. $\beta : * \vdash (\lambda\alpha : *.\alpha \to \alpha)\beta : *$
  3. $\beta : *, x : \beta \vdash (\lambda y : \beta.x) : (\lambda\alpha : *.\alpha \to \alpha)\beta$
  4. $a : *, f : * \to * \vdash f(fa) : *$
  5. $a : * \vdash (\lambda f : * \to *.f(fa)) : (* \to *) \to *$
- В λP верно:
  1. $A : * \vdash (A \to *) : \Box$
  2. Рассмотрим тип A как множество значений типизируемых таким образом и введем $P : A \to *$ Тогда $A : *, P : A \to *, a : A \vdash Pa : *$ Можно рассматривать в таком контексте P как предикат на A. Если для a он возвращает населенный тип, то будем считать это за true, иначе за false. Это теоретико-множественный смысл зависимых типов.

     Можно строить утверждения вида $(\Pi a : A.Pa)$ - для любого a верен предикат P.

- В λw можно задать конъюнкцию, как мы делали еще в системе F. $a\&b = \Pi\gamma : *.(a \to b \to \gamma) \to \gamma$

  Тогда $AND = \lambda a : *.\lambda b : *.a\&b$ $K = \lambda a : *.\lambda b : *.\lambda x : a.\lambda y : b.x$

  $\vdash AND : * \to * \to *$

  $\vdash K : (\Pi a : *.\Pi b : *.a \to b \to a)$

  Тогда получается доказательство того, что из конъюнкции следует первый аргумент!

  $a : *, b : * \vdash (\lambda x : AND\, ab.xa(Kab)) : (AND\, ab \to a) : *$

11 Лекция 13

11.1 Теорема Диаконеску

Теорема 11.1 (Диаконеску)**.** В аксиомах ZF аксиома выбора влечет закон исключенного третьего.

Доказательство. По аксиоме выделения для любого утверждения P мы можем построить два множества из множества $\{0, 1\}$:

$$A = \{x \in \{0,1\} \mid (x = 0) \vee P\} \qquad B = \{x \in \{0,1\} \mid (x = 1) \vee P\}$$

По аксиоме выбора мы знаем, что их декартово произведение непусто. Иначе говоря, существует функция $f : \{A, B\} \to \{0,1\}$, что

$$f(A) \in A \& f(B) \in B$$

Это, по определению двух множеств, эквивалентно

$$(f(A) = 0 \vee P) \,\&\, (f(B) = 1 \vee P)$$

Из этого следует, что

$$(f(A) \neq f(B)) \vee P \tag{$*$}$$

Однако, по принципу объёмности $P \to (A = B)$. Значит, $P \to (f(A) = f(B))$. Значит,

$$(f(A) \neq f(B)) \to \neg P \tag{$**$}$$

Из $*$ и $**$ можно вывести $P \vee \neg P$.

□

Важным следствием данной теоремы является то, что мы не можем воспринимать типы как множества, так как системы типов изоморфны интуиционистской логике в которой нет закона исключенного третьего.