Введение в Теорию Типов
Конспект лекций

Штукенберг Д. Г.
Университет ИТМО

10 ноября 2018 г.

1 Введение

Эти лекции были рассказаны студентам групп M3336–M3339 в 2018 году в Университете ИТМО, на Кафедре компьютерных технологий Факультета информационных технологий и программирования.

Конспект подготовили студенты Кафедры: Егор Галкин (лекции 1 и 2), Илья Кокорин (лекции 3 и 4), Никита Дугинец (лекции 5 и 6), Степан Прудников (лекции 7 и 8).

(возможно, история сложнее)

2 Лекция 3

2.1 Y-комбинатор

Определение 2.1. Комбинатором называется λ-выражение, не имеющее свободных переменных

Определение 2.2. (Y-комбинатор)

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

Очевидно, Y-комбинатор является комбинатором.

Теорема 2.1. $Yf =_\beta f(Yf)$

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 точки для бестипового лямбда-ичисления

Теорема 2.2. В лямбда-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. $\square$

2.2 Рекурсия

С помощью Y-комбинатора можо 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 соответсувует нашим представлениям о рекурсии.)

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

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

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 $(Y\ fact'\ \overline{0})$.

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

Таким образом,

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

2.3 Парадокс Карри

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

Будем тредовать 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 так же 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.

2.4 Импликационный фрагмент интуиционистского исчисления высказываний

Рассмотрим подмножество ИИВ, со следующей грамматикой:

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

То есть состоящее только из мерменных 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 $\vdash \varphi \to \psi \to \varphi$

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

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

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

2.5 Просто типизированное по Карри лямбда-исчисление

Определение 2.3. Тип в просто типизированном лямбда-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 2.4. Язык просто типизированного лямбда-исчисления это язык бестипового лямбда-исчисления.

Определение 2.5. Контекст Γ это список выражений вида $A : \theta$, где A - лямбда-терм, а θ - тип

Определение 2.6. Просто типизипрованное лямбда-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 этихъ двух правил и одной схемы аксиом, то будем говорить, что оно типизируется по Карри.

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

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

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

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

2.6 Отсутствие типа у Y-комбинатора

Теорема 2.3. Y-комбинатор не типизируется в просто типизированном по Карри лямбда 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 типизированна она может быть только по правилу абстракции. Значит, в выводе типа Y-комбинатора есть такой вывод:

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

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.

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

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

2.7 Изоморфизм Карри-Ховарда

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

Просто типизирпованное 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 лямбда-исчисления.

Просто типизирпованное 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

3 Лекция 4

3.1 Расширение просто типизированного λ-исчисления до изоморфного ИИВ

Заметим, что между просто типизированным по Карри λ-исчислением и имликационным 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 $A\&B$, соответсвующий логическому "И"
3. Алгебраический тип $A|B$, соттветсвующий логическому "ИЛИ"

Тип $\perp$ Введём тип $\perp$, сотттветствующий 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 типизированно так же 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) = 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 иъ 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:

$$\frac{\Gamma \vdash < A, B >: \varphi \& \psi}{\Gamma \vdash \Pi_1 < M, N >: \varphi}$$

3. Правило типизации второго проектора:

$$\frac{\Gamma \vdash < A, B >: \varphi \& \psi}{\Gamma \vdash \Pi_2 < M, N >: \psi}$$

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-mathcing, 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_1 - 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:

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

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

Заметим точное соответствие только что введённых конструкций аксиомам ИИВ.

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

3.3 Просто типизированное по Чёрчу λ-исчисление

Определение 3.1. Тип в просто типизированном по Чёрчу λ-исчислении это то же самое, что тип в просто типизированном по Карри λ-исчислении

Определение 3.2. Язык просто типизированного по Чёрчу λ-исчисления удовлетворяет следующей грамматике

$\Lambda_ч ::= x \mid \Lambda_ч\ \Lambda_ч \mid \lambda\ x^\tau.\ \Lambda_ч \mid (\Lambda_ч)$

Замечание 3.1. Иногда абстракция записывается не как $\lambda\ x^\tau.\ \Lambda_ч$, а как $\lambda\ x : \tau.\ \Lambda_ч$

Определение 3.3. Просто типизипрованное 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 этихъ 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 3.1 (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 $\theta = \tau$

3.4 Связь типизации по Чёрчу и по Карри

Определение 3.4 (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 3.2. Пусть $M, N \in \Lambda_ч, M \to_\beta N$, тогда $|M| \to_\beta |N|$

Лемма 3.3. 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 3.4 (Теорема о поднятии)**.**

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 $\theta = \tau$