
STLC 开始的类型论入门

∞ -type Café 暑期学校讲座

Alias Qli

July 02, 2023

什么是类型论

狭义类型论: STLC, MLTT, HoTT 等. (一个类型论)

广义类型论: 这些类型论的汇总, 一个学科.

lambda 演算的动机

函数定义: 解析式

$$f(x) := M$$

注意: 并不存在“解析式不是本质的, 集合表示才是”. 类型论里面解析式就是本质的.

x 只在 M 里面有定义:

$$\frac{x \vdash M}{\lambda x. M}$$

lambda 演算的动机

函数定义: 解析式

$$x \mapsto M$$

注意: 并不存在“解析式不是本质的, 集合表示才是”. 类型论里面解析式就是本质的.

x 只在 M 里面有定义:

$$\frac{x \vdash M}{\lambda x. M}$$

lambda 演算的动机

函数定义: 解析式

$$\lambda x. M$$

注意: 并不存在“解析式不是本质的, 集合表示才是”. 类型论里面解析式就是本质的.

x 只在 M 里面有定义:

$$\frac{x \vdash M}{\lambda x. M}$$

函数应用: 代入

$$f(x) := M$$

$$f(3): \quad x = 3 \text{ 代入 } M$$

$$f(3) = M[x \mapsto 3]$$

函数应用: 代入

$$f(x) := M$$

$f(3)$: 将 M 中的 x 替换成 3

$$f(3) = M[x \mapsto 3]$$

简单类型 lambda 演算

Simply Typed Lambda Calculus

简称 STLC

顾名思义, 简单

相继式演算

$$\frac{\text{Premise}_1 \quad \text{Premise}_2 \quad \dots \quad \text{Premise}_n}{\text{Conclusion}} (\text{Name})$$

Judgement $\begin{cases} \text{Premise} \\ \text{Conclusion} \end{cases}$ 是关于类型论中对象的命题

类型

什么是类型?

引入判断 $A \text{ type}$ 表示 “ A 是类型”.

首先取一个基本类型的集合 $\mathcal{B} = \{\top, \perp, \text{Ans}\}$.

$$\frac{A \in \mathcal{B}}{A \text{ type}} \text{Base}$$

$$\frac{A \text{ type} \quad B \text{ type}}{A \rightarrow B \text{ type}} \text{Func}$$

$\top : \text{top}$

$\perp : \text{bottom}$

语境

语境表示“哪些变量有定义, 类型是什么”.

引入判断 $\Gamma \text{ ctx}$ 表示“ Γ 是语境”.

$$\frac{}{\emptyset \text{ ctx}}$$

$$\frac{\Gamma \text{ ctx}}{\Gamma, x : A \text{ ctx}}$$

“变量 : 类型”二元组组成的反向列表. \emptyset 常常省略.

谓词 $x : A \in \Gamma$ 表示“ x 是 Γ 中类型为 A 的变量”.

项

项: 类型所描述的东西, 那些具有类型的东西.

讨论项不能脱离类型和语境: 引入判断 $\Gamma \vdash M : A$ 表示 “ M 是语境 Γ 下类型为 A 的项”.

取一个常量的集合 $\mathcal{C} = \{\text{tt}, \text{yes}, \text{no}\}$.

$$\overline{\Gamma \vdash \text{tt} : \top}$$

$$\overline{\Gamma \vdash \text{yes} : \text{Ans}} \quad \overline{\Gamma \vdash \text{no} : \text{Ans}}$$

真正重要的规则们:

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A} \text{Var}$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda(x : A). M : A \rightarrow B} \text{Lam}$$

$$\frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B} \text{App}$$

多元函数:

规定 \rightarrow 左结合: $A \rightarrow B \rightarrow C = (A \rightarrow B) \rightarrow C$

函数应用右结合: $MNR = (MN)R$

$$\frac{\frac{f : A \rightarrow B \rightarrow C \quad a : A \quad b : B}{\dots}}{f \ a \ b : C}$$

$$f := \lambda x. \lambda y. \dots = \lambda x \ y. \dots$$

检查一个项的每一部分是用哪条规则构造出来的称为类型检查 (得名于计算机实现).

$$\begin{array}{c}
 \frac{f : A \rightarrow B \in f : A \rightarrow B, x : A}{f : A \rightarrow B, x : A \vdash f : A \rightarrow B} \text{Var} \quad \frac{x : A \in f : A \rightarrow B, x : A}{f : A \rightarrow B, x : A \vdash x : A} \text{Var} \\
 \hline
 \frac{\quad}{f : A \rightarrow B, x : A \vdash f x : B} \text{App} \\
 \hline
 \frac{\quad}{f : A \rightarrow B \vdash \lambda x. f x : A \rightarrow B} \text{Lam} \\
 \hline
 \frac{\quad}{\emptyset \vdash \lambda f x. f x : (A \rightarrow B) \rightarrow A \rightarrow B} \text{Lam}
 \end{array}$$

替换

替换: 将项 M 中的某个变量 x 全部换成另一个项 N , 记作 $M[x \mapsto N]$.

注意: N 里面有个变量 M 正在用, 怎么办?

例子: $(\lambda x. y)[y \mapsto x] \stackrel{?}{=} \lambda x. x$, 能替换吗?

解决方法: 给 x 改名字: $\lambda x. y = \lambda z. y, (\lambda z. y)[y \mapsto x] = \lambda z. x$

给函数参数和它的所有使用改名字, 只要这样不造成新的重名: alpha 转换. 我们认为转换前后的项等价.

总可以通过有限步 alpha 转换使不同的变量两两不重名.

因此我们可以默认所有变量两两不重名: 替换可以直接换.

归约

归约就是化简. 归约的手段是两条规则, 归约的目标是既约形式.

$$\frac{x : A \vdash M : B \quad N : A}{(\lambda x. M)N \equiv M[x \mapsto N] : B} \beta \text{ rule}$$

$$\frac{f : A \rightarrow B}{f \equiv \lambda x. f x : A \rightarrow B} \eta \text{ rule}$$

例子:

$$\begin{aligned} & (\lambda(f : \text{Ans} \rightarrow \text{Ans}). f \text{ yes})(\lambda(x : \text{Ans}). x) \\ \equiv & (f \text{ yes})[f \mapsto \lambda(x : \text{Ans}). x] && (\beta \text{ reduction}) \\ \equiv & (\lambda(x : \text{Ans}). x) \text{ yes} && (\text{substitution}) \\ \equiv & x[x \mapsto \text{yes}] && (\beta \text{ reduction}) \\ \equiv & \text{yes} && (\text{substitution}) \end{aligned}$$

总结

- 类型讨论的是**语境中有类型的项**
- 一个项是不是良类型的?

尝试用项的规则构造它/类型检查

- 一个项的最简形式是什么?

归约.

方法? beta/eta 规则

STLC 的模型

模型, 也叫语义, 在计算机里面叫指称语义, 近似于解释器.

思想: 为理论中的每一个东西在模型中找一个对应 (映射, 解释), 使得原本成立的性质都成立 (模型中可以有多出来的性质).

在理论中成立的在每个模型中都成立; 在某个模型里不成立的在理论中不成立.

注意: 命题不成立和命题的否定成立是两个概念, 后者能推出前者.

STLC 的模型: 将类型, 语境, 项分别映射到模型, 使得 alpha beta eta 规则在模型里也成立. 惯例: 用同一个记号 $\llbracket - \rrbracket$ 表示这三个映射.

STLC 的集合模型: 将大家都解释到集合. 对各自的结构归纳定义.

类型:

$$\llbracket \perp \rrbracket = \emptyset$$

$$\llbracket \top \rrbracket = \{*\}$$

$$\llbracket \text{Ans} \rrbracket = \{0, 1\}$$

$$\llbracket A \rightarrow B \rrbracket = \llbracket A \rrbracket \rightarrow \llbracket B \rrbracket$$

语境:

$$\llbracket \emptyset \rrbracket = \{*\}$$

$$\llbracket \Gamma, x : A \rrbracket = \llbracket \Gamma \rrbracket \times \llbracket A \rrbracket$$

项 $\Gamma \vdash M : A$ 解释到 $\llbracket \Gamma \rrbracket \rightarrow \llbracket A \rrbracket$ 的函数.

$$\llbracket \Gamma \vdash \text{tt} : \top \rrbracket = - \mapsto *$$

$$\llbracket \Gamma \vdash \text{yes} : \text{Ans} \rrbracket = - \mapsto 1$$

$$\llbracket \Gamma \vdash \text{no} : \text{Ans} \rrbracket = - \mapsto 0$$

$$\llbracket \Gamma \vdash x : A \rrbracket = \pi_{i+1} \quad (x : A \text{ 是 } \Gamma \text{ 中的第 } i \text{ 项})$$

$$\frac{\llbracket \Gamma \vdash M : A \rightarrow B \rrbracket = m \quad \llbracket \Gamma \vdash N : A \rrbracket = n}{\llbracket \Gamma \vdash MN : B \rrbracket = \gamma \mapsto m(\gamma)(n(\gamma))}$$

$$\frac{\llbracket \Gamma, x : A \vdash M : B \rrbracket = m}{\llbracket \Gamma \vdash \lambda x. M : A \rightarrow B \rrbracket = \gamma \mapsto (x \mapsto m(\gamma, x))}$$

集合模型能证明一些简单的推论.

更复杂的模型能证明更复杂的推论!

模型可以照搬到代码里当解释器.

Bool 类型

Ans 类型的项没法使用: 不存在从 Ans 类型出发的非平凡的常函数.
缺少消去规则!

- 如何构造一个类型的项: 引入规则
- 如何使用一个类型的项: 消去规则

形成规则:

$$\overline{\mathbb{B} \text{ type}}$$

引入规则:

$$\overline{\text{true} : \mathbb{B}} \quad \overline{\text{false} : \mathbb{B}}$$

消去规则:

$$\frac{b : \mathbb{B} \quad c_t : C \quad c_f : C}{\text{elim}_{\mathbb{B}}(b, c_t, c_f) : C}$$

计算规则:

$$\frac{c_t : C \quad c_f : C}{\text{elim}_{\mathbb{B}}(\text{true}, c_t, c_f) \equiv c_t : C} \quad \frac{c_t : C \quad c_f : C}{\text{elim}_{\mathbb{B}}(\text{false}, c_t, c_f) \equiv c_f : C}$$

计算规则描述了消去规则如何抵消引入规则, 反过来, 引入规则如何抵消消去规则呢?

唯一性规则:

$$\frac{b : \mathbb{B} \quad c : \mathbb{B} \rightarrow C}{\text{elim}_{\mathbb{B}}(b, c \text{ true}, c \text{ false}) \equiv c \ b : C}$$

积类型

元组, 笛卡尔积

形成规则:

$$\frac{A \text{ type} \quad B \text{ type}}{A \times B \text{ type}}$$

引入规则:

$$\frac{a : A \quad b : B}{(a, b) : A \times B}$$

消去规则:

$$\frac{p : A \times B}{\text{fst}(p) : A} \quad \frac{p : A \times B}{\text{snd}(p) : B}$$

计算规则:

$$\frac{a : A \quad b : B}{\text{fst}((a, b)) \equiv a : A} \quad \frac{a : A \quad b : B}{\text{snd}((a, b)) \equiv b : B}$$

唯一性规则:

$$\frac{p : A \times B}{p \equiv (\text{fst}(p), \text{snd}(p)) : A \times B}$$

余积类型

和类型, 具名联合. 积类型的对偶.

形成规则:

$$\frac{A \text{ type} \quad B \text{ type}}{A + B \text{ type}}$$

引入规则:

$$\frac{a : A}{\text{inl}(a) : A + B} \quad \frac{b : B}{\text{inr}(b) : A + B}$$

消去规则:

$$\frac{s : A + B \quad c_l : A \rightarrow C \quad c_r : B \rightarrow C}{\text{elim}_{A+B}(s, c_l, c_r) : C}$$

自然数类型

一个递归的类型!

引入规则:

$$\frac{}{\text{zero} : \mathbb{N}} \quad \frac{n : \mathbb{N}}{\text{suc}(n) : \mathbb{N}}$$

消去规则:

$$\frac{n : \mathbb{N} \quad c_0 : C \quad c_s : \mathbb{N} \rightarrow C \rightarrow C}{\text{elim}_{\mathbb{N}}(n, c_0, c_s) : C}$$

计算规则:

$$\frac{c_0 : C \quad c_s : \mathbb{N} \rightarrow C \rightarrow C}{\text{elim}_{\mathbb{N}}(\text{zero}, c_0, c_s) \equiv c_0 : C}$$

$$\frac{n : \mathbb{N} \quad c_0 : C \quad c_s : \mathbb{N} \rightarrow C \rightarrow C}{\text{elim}_{\mathbb{N}}(\text{suc}(n), c_0, c_s) \equiv c_s \ n \ \text{elim}_{\mathbb{N}}(n, c_0, c_s) : C}$$

$$\text{elim}_{\mathbb{N}}(n, c_0, c_s) \equiv \underbrace{c_s(n-1)(c_s(n-2)\dots(c_s \ \text{zero} \ c_0))}_{n \text{ times}}$$

模式匹配

消去规则的另一种语法: 对引入规则分情况讨论

$\text{elim}_{\mathbb{B}}(b, c_t, c_f)$	case b of
	$\text{true} \rightarrow c_t$
	$\text{false} \rightarrow c_f$

$f \text{ fst}(p) \text{ snd}(p)$	case p of
	$(a, b) \rightarrow f a b$

$\text{elim}_{A+B}(s, c_l, c_r)$	case s of
	$\text{inl}(a) \rightarrow c_l a$
	$\text{inr}(b) \rightarrow c_r b$

自然数的模式匹配

$$\begin{aligned}
 f &:= \lambda n. \text{elim}_{\text{Nat}}(n, c_0, c_s) & f \text{ zero} &:= c_0 \\
 & & f \text{ suc}(m) &:= c_s \ m \ (f \ m)
 \end{aligned}$$

定义自然数加法试试看:

$$\begin{aligned}
 \text{add} &: \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N} \\
 \text{add } m \ n &:= \text{elim}_{\mathbb{N}}(m, n, \lambda_. \lambda n. \text{suc}(n)) \\
 \text{add zero } n &:= n \\
 \text{add suc}(m) \ n &:= \text{suc}(\text{add } m \ n)
 \end{aligned}$$

我们允许任意递归吗? 停机性检查

函数类型的 5 种规则

形成规则:

$$\frac{A \text{ type} \quad B \text{ type}}{A \rightarrow B \text{ type}}$$

引入规则:

$$\frac{x : A \vdash M : B}{\lambda x. M : A \rightarrow B} \text{Lam}$$

消去规则:

$$\frac{M : A \rightarrow B \quad N : A}{MN : B} \text{App}$$

计算规则:

$$\frac{x : A \vdash M : B \quad N : A}{(\lambda x. M)N \equiv M[x \mapsto N] : B} \beta$$

唯一性规则:

$$\frac{f : A \rightarrow B}{\lambda x. f x \equiv f : A \rightarrow B} \eta$$

这些规则就是我们对于函数的要求.

特殊的类型

太简单了, 没有完整的 4 种规则

\top 只有引入规则和计算规则:

$$\frac{}{tt : \top} \quad \frac{e : \top}{e \equiv tt : \top}$$

\perp 只有消去规则和唯一性规则:

$$\frac{e : \perp}{\text{elim}_{\perp}(e) : C} \quad \frac{e : \perp \quad c : C}{\text{elim}_{\perp}(e) \equiv c : C}$$

Curry-Howard 同构

直觉主义自然演绎: 不包含排中律的命题逻辑系统.

记 Γ 为一个命题的有限集合, 称为假设 (或公理). 定义仅在下列情况中, 命题 P 可由 Γ 推出, 记为 $\Gamma \vdash P$:

- P 是 Γ 中的一个命题: 引入假设或使用公理

$$\frac{P \in \Gamma}{\Gamma \vdash P} \text{Ax}$$

- P 具有 $A \rightarrow B$ 的形式, 并且 $\Gamma, A \vdash B$: 演绎定理

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \rightarrow I$$

$$\frac{P \in \Gamma}{\Gamma \vdash P} \text{Ax}$$

- 存在 A 使得有 $\Gamma \vdash A \rightarrow P$ 和 $\Gamma \vdash A$: 肯定前件

$$\frac{\Gamma \vdash A \rightarrow P \quad \Gamma \vdash A}{\Gamma \vdash P} \rightarrow E$$

这东西怎么和 Var, Lam, App 这么像? 不是偶然

- 函数类型对应蕴含
- 类型对应命题
- 项对应证明

类型论就对应逻辑系统. 这就是 CH 同构.

证明 $(P \rightarrow Q) \rightarrow (Q \rightarrow R) \rightarrow P \rightarrow R$:

$\text{proof} : (P \rightarrow Q) \rightarrow (Q \rightarrow R) \rightarrow P \rightarrow R$

$\text{proof} := \lambda f\ g\ x. g\ (f\ x)$

这个对应能推得更广:

- \top 对应真命题, tt 是它的证明.
- \perp 对应假命题, elim_\perp 是谎言爆炸原理.
- $A \times B$ 对应 $A \wedge B$: 如果命题 A 有证明 a , 命题 B 有证明 b , 那么 $A \wedge B$ 有证明 (a, b) .
- $A + B$ 对应 $A \vee B$: 如果命题 A 有证明 a , 那么 $A \vee B$ 有证明 $\text{inl}(a)$; 如果命题 B 有证明 b , 那么 $A \vee B$ 有证明 $\text{inr}(b)$.

“ P 的否定” 就是 “ P 能推出假”.

$A \rightarrow B \rightarrow A + B$ 有两个项, 对应两个证明?

是的

更高级的类型论里面能够精确表达“命题”的概念

直觉主义逻辑

直觉主义逻辑讲究构造

还能做数学吗? 能

Godel 嵌入

总结

- 研究类型论性质的方法: 构造模型
- 更多类型: Bool 类型, 积类型, 余积类型, 自然数类型
- 形成规则, 引入规则, 消去规则 (模式匹配), 计算规则, 唯一性规则
- CH 同构, 直觉主义