pedagogy

教程 : 类型论 [tutorial-type-theory]

SimplicialCat

本文是杨家同于 2026 年 1 月在求真书院形式化数学冬令营上的讲座的笔记, 加入了个人的见解. 假想的读者是有一定数学成熟度但对类型论没有了解的同学.

什么是类型论

类型论是一种不同于集合论的做数学的方法, 因其在处理逻辑问题上的优越性, 而在计算机形式化数学这一领域得到广泛应用. 类型论的首要特征是**给每个对象一个类型**. 常量, 变量, 函数, 命题, 命题的证明, ... 每个对象有唯一且可计算的类型; 甚至 (在某些类型论中) 一个类型也有它的类型. **命题是类型**的原则让逻辑学成为类型论的一个子集.

与之相比, 集合论中每个对象的类型都是 "集合", 这种理论上的简洁造成了实操上的混乱. 例如用集合论定义自然数, $1 = \{\varnothing\}$, 而 $2 = \{\varnothing, \{\varnothing\}\}$, 我们便可写出 $1 \in 2$ 这样正确但毫无意义的话. 这句话在类型论中是不允许的, 因为 $1, 2$ 只是类型 $\mathbb{N}$ 的元素, 它们之间无法谈论 "$\in$" 这个关系. 懂得道理的人类可以自觉避免说出无意义的话, 那么如何让死板的机器也能永远做正确的事呢? 最好的办法就是明确地记录每个对象的身份, 用一套成体系的算法计算出每个构造的新对象的类型; 这样, 机器就能确保它说的一句话是有**数学意义**的.

其实, 类型论的思想已经隐约地存在于每一位数学家的心中. 我们做的事情只是把它明确地表达出来.

类型的三要素

设想有两个类型 A, B, 我们希望定义一个类型 $A \times B$, 称为**积类型** (product type). 这个类型的一个元素相当于 A 的一个元素和 B 的一个元素, 也即一个二元组 (a, b). 类型论期待我们用如下的方式刻画这个类型:

1. 如何**创建**这个类型, 称为**形成法则** (formation rule); 这里, 由两个已知类型 A, B 可以形成这个类型 $A \times B$.
2. 如何**引入**这个类型的元素, 称为**引入法则** (introduction rule); 这里, 由类型 A 的元素 a, 类型 B 的元素 b, 可以得到类型 $A \times B$ 的元素 (a, b).
3. 如何**使用**这个类型的元素, 称为**消去法则** (elimination rule); 这里, 由类型 $A \times B$ 的元素 p, 可以拆出来类型 A 的元素 $\mathrm{pr}_1(p)$ 和类型 B 的元素 $\mathrm{pr}_2(p)$.
4. 作为对消去法则的补充, 还需要我们规定**消去法则如何作用于引入法则**, 又称**计算法则**; 这里, $\mathrm{pr}_1((a, b)) = a, \mathrm{pr}_2((a, b)) = b$.

类型论中的所有类型都是用这样三个要素刻画的.

下一个例子是 "**和类型**" (sum type) $A + B$.

1. 形成法则: 由两个类型 A, B 可构造和类型 $A + B$.
2. 引入法则: 由 $a : A$ 可得 $L(a) : A + B$, 由 $b : B$ 可得 $R(b) : A + B$.

3. 消去法则: 由 $f : A \to C, g : B \to C$ 可得 $(f, g) : A + B \to C$.
4. 计算法则: $(f, g)(L(a)) = f(a), (f, g)(R(b)) = g(b)$.

和类型的消去法则体现了范畴论中的和的泛性质.

函数类型 (function type) $A \to B$.

1. 形成法则: 由两个类型 A, B 可构造函数类型 $A \to B$.
2. 引入法则: 由含一个变量 $x : A$ 的表达式 $\Phi(x) : B$, 可得 $(x \mapsto \Phi(x)) : A \to B$.
3. 消去法则: 由 $a : A, f : A \to B$, 可得 $f(a) : B$.
4. 计算法则: 对于 $a : A, f = (x \mapsto \Phi(x)) : A \to B$, 有 $f(a) = \Phi(a)$. 此外, $(x \mapsto f(x))$ 和 f 是一个意思.

空类型 (empty type).

1. 形成法则: 我们规定有一个类型 $\mathbf{0}$, 称为空类型.
2. 引入法则: 没有任何方法可以手动引入 $\mathbf{0}$ 的元素.
3. 消去法则: 由 $x : \mathbf{0}$, 可得**任何**类型 C 的元素 $\mathsf{elim}_{\mathbf{0}}(x) : C$. 这就像是空集到任何集合都有一个映射.

单元类型 (unit type).

1. 形成法则: 我们规定有一个类型 $\mathbf{1}$, 称为单元类型.
2. 引入法则: 有一个元素 $\star : \mathbf{1}$.
3. 消去法则: 对 $a : A$, 有一个函数 $f : \mathbf{1} \to A, f(\star) = a$.

类似地, 可定义有两个元素的类型 $\mathbf{2}$.

一种重要的定义类型的模式称为**归纳类型** (inductive type), 以自然数类型为例. (其实空类型, 单元类型这些也是归纳类型, 不过对初学者可能有点烧脑, 还是从熟悉的自然数开始介绍.)

1. 形成法则. 我们规定有一个类型 $\mathbb{N}$.
2. 引入法则. 有一个元素 $0 : \mathbb{N}$; 对任意 $n : \mathbb{N}$ 有一个元素 $n' : \mathbb{N}$, 称为 n 的**后继**.
3. 消去法则 (又称**归纳**). 对任意类型 C, 要给出一个函数 $f : \mathbb{N} \to C$, 只要给出 $f(0) : C$, 以及 $f(n') : C$ 的表达式, 其中 $f(n') : C$ 的表达式**可以使用前一项** $f(n)$.

例如, 要定义 $\mathrm{Even} : \mathbb{N} \to \mathbf{2}$, 只需给出 $\mathrm{Even}(0) = 1, \mathrm{Even}(1) = 0, \mathrm{Even}(n'') = \mathrm{Even}(n)$.

归纳的一种理解方式是, 在消去法则中, 当我们要使用一个元素时, **不妨设**它具有某种形式 (来自某个引入法则). 例如要使用一个自然数 n, 不妨设它具有两种形式之一: 或者是 0, 或者是某个自然数 n 的后继. 后面我们将更严格地讲解归纳类型.

类型的类型: 宇宙

一个类型也有它的类型. 粗略地说, 所有类型都是某个类型 $\mathcal{U}$ 的元素, 称之为**宇宙** (universe). 但假若 $\mathcal{U}$ 也是 $\mathcal{U}$ 的元素会导致矛盾; 实践上人们的做法是规定一串宇宙 $\mathcal{U}_1, \mathcal{U}_2, \cdots$, "普通" 的类型是宇宙 $\mathcal{U}_1$ 的元素, 而 $\mathcal{U}_1$ 是 $\mathcal{U}_2$ 的元素, 以至于无穷.

但在多数情况下, 我们不需要明确所处的宇宙层级, 而是以笼统的 $\mathcal{U}$ 代替.

比通常宇宙低一个层级的宇宙是**命题**的类型 $\mathsf{Prop} = \mathcal{U}_0$. 所谓**命题**是满足如下条件的类型 P: 对任意 $x, y : P$ 有 $x = y$. 在 Curry–Howard 对应之下, 这意味着一个命题的任意两个证明都视为相等.

依值类型: 到宇宙的函数

宇宙 $\mathcal{U}$ 也是 (某个更大的宇宙的) 类型, 从而我们可以谈论类型 A 到 $\mathcal{U}$ 的函数 $B : A \to \mathcal{U}$, 称为**依值类型** (dependent type). 它对每个 $a : A$ 都指定了一个类型 $B(a)$.

例如, 固定 $A : \mathcal{U}$, 对每个自然数 $n : \mathbb{N}$ 有一个类型 A^n, 那么这些类型合起来构成了一个依值类型 $V_A : \mathbb{N} \to \mathcal{U}$. 这个依值类型是通过归纳定义的: $V_A(0) = \mathbf{1}$, $V_A(n') = V_A(n) \times A$.

写给有相关背景的读者: 依值类型是代数拓扑中的纤维化 (fibration) (纤维丛) 或代数几何中**相对概形** (relative scheme) 的类比. 准确地说, 它们都是纤维范畴的例子, 而纤维范畴可以作为依值类型的范畴语义.

对一个依值类型 $B : A \to \mathcal{U}$ 我们可以做两件事: 把每个类型 "加起来", 叫做**依值和** (dependent sum)

$$\sum_{a:A} B(a),$$

以及把每个类型 "乘起来", 叫做**依值积** (dependent product)

$$\prod_{a:A} B(a).$$

两者分别是乘积类型和函数类型的推广, 因为当 $B(a)$ 不依赖于 a 时, $\sum_{a:A} B(a), \prod_{a:A} B(a)$ 分别退化为 $A \times B, A \to B$. 因此, 有些文献或形式化系统中也将 $\sum_{a:A} B(a)$ 记为 $(a : A) \times B(a)$, 将 $\prod_{a:A} B(a)$ 记为 $(a : A) \to B(a)$.

依值积类型的元素又称为**依值函数**. 归纳类型 (如 $\mathbb{N}$) 的消去法则可以升级为依值函数的构造法则, 以 $\mathbb{N}$ 为例: 要给出一个依值函数 $f : (n : \mathbb{N}) \to C(n)$, 只要给出 $f(0) : C(0)$, 以及 $f(n') : C(n')$ 的表达式, 依赖于 $f(n) : C(n)$, 这相当于给出函数 $C(n) \to C(n')$.

如果依值类型类比于纤维丛, 那么依值和类比于纤维丛的**全空间**, 而依值积类比于纤维丛的截面空间.

写给有范畴论背景的读者: 依值和类型与依值积类型的范畴语义分别是拉回的**左伴随**和**右伴随**.

类型 A 到命题宇宙的函数 $P : A \to \mathcal{U}_0$ 是类型 A 上的**谓词** (predicate), 也可以理解为 A 的 "子集". 换言之, 子集是一种特殊的依值类型, 其中每个类型都是命题. 对于谓词 $P : A \to \mathcal{U}_0$, 其对应的**子类型**是依值和 $\sum_{a:A} P(a)$, 带有到 A 的 "投影" (或称 "含入") 映射.

命题即类型: Curry–Howard 对应

类型论不仅能让机器保证它说出的话**有意义**, 而且能让它保证作出的证明是**对**的 (而这当然是我们做形式化数学最关心的事情). 这是因为 "有意义" 和 "对" 根本是一回事: 由于**命题即类型** (Propositions as Types) 的原则, 在类型论中检查一个元素的类型是否正确, 和检查一个证明是否正确, 做的是同一件事.

比如, 检查 (a, b) 是类型 $A \times B$ 的元素, 就是分别检查 a 是 A 的元素, 以及 b 是 B 的元素. 与之对应地, 检查 (p, q) 构成命题 $P \wedge Q$ 的证明, 就是检查 p 构成 P 的证明, 以及 q 构成 Q 的证明.

类型论	逻辑
$a : A$	a 是 A 的证明
$A \times B$	A 且 B
$A + B$	A 或 B
$A \to B$	A 蕴涵 B
$\mathbf{0}$	假
$\mathbf{1}$	真
$\sum_{a:A} B(a)$	存在 a 使得 $B(a)$
$\prod_{a:A} B(a)$	对任意 a 有 $B(a)$

直觉主义逻辑

不是每个经典逻辑的永真命题都能通过 Curry–Howard 对应得到证明. 一个重要的例子是排中律 (law of excluded middle)

$$A + \neg A = A + (A \to \mathbf{0}).$$

要证明这个命题 (构造这个类型的元素), 我们只能要么构造 A 的元素, 要么构造映射 $A \to \mathbf{0}$. 但是当我们不知道 A 的更多信息时, 无论哪一边都没有办法构造, 进退两难. 类似地, **双重否定律** (law of double negation)

$$\neg\neg A \to A$$

也不能证明. 在需要经典逻辑的类型论中, 这些定律是作为公理提供的. 所谓公理就是人为规定某个特定类型的一个元素.

Backlinks

Meta. 教学 [pedagogy]
SimplicialCat

subject类型论 [类型论]
SimplicialCat