

# 二阶 (second order) 类型化的 $\lambda$ -演算

读书笔记

许博

## 1 疑惑

1. 所谓依赖是否只存在于抽象, 如  $\lambda x : \sigma.M$  依赖于  $x$ , 而  $MN$  并不依赖于  $M$  或  $N$ ?

## 2 类型抽象和类型应用

在  $\lambda \rightarrow$  中, 遇到的抽象和应用都是在项层面, 也即所谓的一阶:

- 如在抽象过程中, 项  $M$  上  $x$  的抽象, 其中假设  $x : \sigma$ ,  $\lambda x : \sigma.M$  依赖于项  $x$ 。

因此, 在  $\lambda \rightarrow$  中, 可以构建依赖于项的项 (terms depending on terms)。

- 对应于抽象的是应用, 项  $MN$  是应用项  $M$  到项  $N$  的结果。

$\lambda \rightarrow$  中的抽象是一阶抽象, 或一阶依赖, 因为抽象是在项上进行的。应用也是一阶的。

本章将引入依赖于类型的项 (terms depending on types), 相关的运算称之为二阶运算或二阶依赖。而所获得的系统称为二阶类型化的  $\lambda$ -演算, 记为  $\lambda 2$ 。

从一个例子, 来看  $\lambda \rightarrow$  表达能力带来的限制, 比如对于接收一个输入然后直接返回输入自身的恒等函数:

- 对于自然数, `nat`, 恒等函数为  $\lambda x : \text{nat}.x$ 。
- 对于布尔值, `bool`, 它是  $\lambda x : \text{bool}.x$ 。

...

对于每一个类型，都有一个与之对应的恒等函数，而在  $\lambda \rightarrow$  中，通用的恒等函数无法表示，尽管对于所有类型，它的行为是相同的。为了表示通用的（恒等）函数，加入另一种抽象：

$$\lambda\alpha : *. \lambda x : \alpha. x。$$

新加入的抽象是出现在第一个  $\lambda$  之后的类型变量  $\alpha$ ，而符号  $*$  表示所有  $(\mathbb{T}2)$  类型的类型，换言之，如果  $\alpha$  是一个  $(\mathbb{T}2)$  类型，那么  $\alpha \in *$ 。需要注意的是， $\lambda\alpha : *. \lambda x : \alpha. x$  是一个依赖于类型的项（a term depending on a type），它所依赖的类型是  $\alpha$ 。

得到的这个（二阶）项称为多态的（polymorphic）恒等函数，需要注意的是它自己并非一个恒等函数，而是一个潜在的（potential）恒等函数。在进行（二阶）应用以及  $\beta$ -规约后可以得到一个名副其实（genuine）的恒等函数，如：

- $(\lambda\alpha : *. \lambda x : \alpha. x) \text{nat} \rightarrow_{\beta} \lambda x : \text{nat}. x$ ，得到一个在  $\text{nat}$  上的恒等函数。
- $(\lambda\alpha : *. \lambda x : \alpha. x) (\text{nat} \rightarrow \text{bool}) \rightarrow_{\beta} \lambda x : (\text{nat} \rightarrow \text{bool}). x$ ，得到一个在  $\text{nat} \rightarrow \text{bool}$  上的恒等函数。

因此当以这种方式扩展  $\lambda \rightarrow$  时，需要引入二阶抽象和应用，除此之外还需要对于二阶项的  $\beta$ -规约。

第二个是关于迭代（iteration）的例子，也即一个函数的重复应用。对于一个类型  $\sigma$  和一个具有类型  $\sigma \rightarrow \sigma$  的函数  $F$ ，定义  $D_{\sigma, F}$  为映射类型  $\sigma$  的  $x$  到  $F(F(x))$  的函数。 $D_{\sigma, F}$  是  $F$  的二次迭代（second iteration），也可记为  $F \circ F$ （ $F$  和自己的复合函数）。

在  $\lambda \rightarrow$  中  $D_{\sigma, F}$  可以通过项  $\lambda x : \sigma. F(Fx)$  表示，而若要定义为任意的  $\sigma$  以及任意的  $F : \sigma \rightarrow \sigma$  定义通用的  $D$ ，则需要使用类型变量  $\alpha$  而不是固定的类型  $\sigma$ ，使用满足  $f : \alpha \rightarrow \alpha$  项变量  $f$  而不是固定的函数  $F$ 。通过由  $f$  和  $\alpha$  的抽象可以得到  $D$  的定义：

$$D \equiv \lambda\alpha : *. \lambda f : \alpha \rightarrow \alpha. \lambda x : \alpha. f(fx)。$$

同样可以给出函数符合运算  $\circ$  在  $\lambda 2$  中的定义：

$$\circ \equiv \lambda\alpha : *. \lambda\beta : *. \lambda\gamma : *. \lambda f : \alpha \rightarrow \beta. \lambda g : \beta \rightarrow \gamma. \lambda x : \alpha. g(fx)。$$

### 3 $\Pi$ -类型

$\lambda 2$  中的二阶项，如  $\lambda\alpha : *. \lambda x : \alpha. x$ ，同样具有类型，表示为  $\Pi\alpha : *. \alpha \rightarrow \alpha$ ，也即  $\lambda\alpha : *. \lambda x : \alpha. x : \Pi\alpha : *. \alpha \rightarrow \alpha$ 。其中  $\Pi\alpha : *$  表示  $\alpha$  为绑定（类型）

变量，且它的类型是  $*$ 。

在引入  $\Pi$ -类型之前，等同项  $\lambda\alpha : *. \lambda x : \alpha. x$  和  $\lambda\beta : *. \lambda x : \beta. x$  的类型因为没有标识绑定（类型）变量而不同。

## 4 二阶抽象和应用规则

由于引入了二阶抽象和二阶应用以及  $\Pi$ -类型，因此对于  $\lambda \rightarrow$  的推导系统也要进行相应的扩展。

**定义 4.1**（二阶抽象规则）

$$(abst_2) \frac{\Gamma, \alpha : * \vdash M : A}{\Gamma \vdash \lambda\alpha : *. M : \Pi\alpha : *. A}$$

**定义 4.2**（二阶应用规则）

$$(appl_2) \frac{\Gamma \vdash M : \Pi\alpha : *. A \quad \Gamma \vdash B : *}{\Gamma \vdash MB : A[\alpha := B]}$$

在二阶应用规则中， $B : *$  意为  $B$  是一个类型。

## 5 系统 $\lambda_2$

首先扩展类型的定义，其中  $\mathbb{V}$  是类型变量的集合  $\{\alpha, \beta, \gamma, \dots\}$ ：

$$\mathbb{T}2 = \mathbb{V} | (\mathbb{T}2 \rightarrow \mathbb{T}2) | (\Pi \mathbb{V} : *. \mathbb{T}2)。$$

第二步，扩展预先类型化的  $\lambda$ -项的集合 ( $\Lambda_{\mathbb{T}}$ ) 以允许二阶抽象和应用：

**定义 5.1**（二阶预先类型化的  $\lambda$ -项， $\lambda_2$ -项， $\Lambda_{\mathbb{T}2}$ ）

$$\Lambda_{\mathbb{T}2} = V | (\Lambda_{\mathbb{T}2} \Lambda_{\mathbb{T}2}) | (\Lambda_{\mathbb{T}2} \mathbb{T}2) | (\lambda V : \mathbb{T}2. \Lambda_{\mathbb{T}2}) | (\lambda \mathbb{V} : *. \Lambda_{\mathbb{T}2})。$$

需要注意的是，其中有两类变量：对象（object）变量  $V$  和类型变量  $\mathbb{V}$ 。由对象变量进行一阶抽象，由类型变量进行二阶抽象。同样有与之对应的一阶应用和二阶应用。

**定义 5.2**（语句（*statement*），声明）

(1) 一个语句形如  $M : \sigma$ ，其中  $M \in \Lambda_{\mathbb{T}2}$  且  $\sigma \in \mathbb{T}2$ ，或形如  $\sigma : *$ ，其中  $\sigma \in \mathbb{T}2$ 。

(2) 一个声明是由一个项变量或一个类型变量作为对象的语句。

因为  $\lambda \rightarrow$  中的类型常量在  $\lambda 2$  中变成了类型变量，所以与  $\lambda \rightarrow$  相比， $\lambda 2$  更加严格一些，因为所有的变量使用前必须被声明，在使用变量之前，我们需要知道所有变量的类型。所以所有的抽象的类型变量也需要被声明，这也是抽象的类型变量与前一章中的类型变量不同的地方，文中假设出现的类型变量都是抽象的，而非代表某一具体类型的类型变量。

**定义 5.3** ( $\lambda 2$ -上下文；域； $dom$ )

(1)  $\emptyset$  是一个  $\lambda 2$ -上下文；

$dom(\emptyset) = ()$ ，空的列表。

(2) 如果  $\Gamma$  是一个  $\lambda 2$ -上下文， $\alpha \in \mathbb{V}$  且  $\alpha \notin dom(\Gamma)$ ，则  $\Gamma, \alpha : *$  是一个  $\lambda 2$ -上下文；

$dom(\Gamma, \alpha : *) = (dom(\Gamma), \alpha)$ 。

(3) 如果  $\Gamma$  是一个  $\lambda 2$ -上下文，如果  $\rho \in \mathbb{T}2$  且对于  $\rho$  中出现的所有自由类型变量  $\alpha$  都有  $\alpha \in dom(\Gamma)$ ，以及如果  $x \notin dom(\Gamma)$ ，则  $\Gamma, x : \rho$  是一个  $\lambda 2$ -上下文；

$dom(\Gamma, x : \rho) = (dom(\Gamma), x)$ 。

需要注意的是，定义需要在一个  $\lambda 2$ -上下文中的所有项变量以及类型变量彼此之间都是不同的（符号上）。

为了符合新的上下文定义，修改  $\lambda \rightarrow$  的 ( $var$ )-规则，以能够开始关联一个正确的  $\lambda 2$ -上下文的变量的类型的推导：

**定义 5.4** ( $\lambda 2$  的（变量， $var$ ）规则）

（变量， $var$ ）如果  $\Gamma$  是一个  $\lambda 2$ -上下文且  $x : \sigma \in \Gamma$ ，则  $\Gamma \vdash x : \sigma$ 。

$\lambda \rightarrow$  中的 ( $appl$ ) 规则以及 ( $abst$ ) 规则将继续使用而不进行修改。

需要注意的是，上一节中我们没有机会使用定义的 ( $appl_2$ )-规则，因为其中的第二个 **premiss** 是  $\Gamma \vdash B : *$ ，但是没有规则可以产生这个形式的 **conclusion**，为了解决这个问题，需要添加一条规则：

**定义 5.5** （形成规则，*formation rule*）(*form*) 如果  $\Gamma$  是一个  $\lambda 2$ -上下文， $B \in \mathbb{T}2$  且  $B$  中所有的自由类型变量已经在  $\Gamma$  中声明，则  $\Gamma \vdash B : *$ 。

这个规则告诉了我们一个正确形式化的  $\lambda 2$ -类型  $B$  的类型是什么。

需要注意的是，(*form*)-规则有三个副条件，但是没有 **premises**，因此，它可以像 ( $var$ )-规则一样出现在推导树的叶子。

**定义 5.6** (合法的  $\lambda_2$ -项) 如果存在一个  $\lambda_2$ -上下文  $\Gamma$  和一个类型  $\rho \in \mathbb{T}_2$  且  $\Gamma \vdash M : \rho$ , 则项  $M \in \Lambda_{\mathbb{T}_2}$  称为合法的。

## 6 $\lambda_2$ 的性质

调整  $\alpha$ -变换的定义, 以适应  $\Pi$ -类型:

**定义 6.1** ( $\alpha$ -变换或  $\alpha$ -等价, 扩展的)

(1a) (项变量的重命名)

如果  $y \notin FV(M)$  且  $y$  不作为绑定变量在  $M$  中出现, 则  $\lambda x : \sigma.M =_\alpha \lambda y : \sigma.M^{x \rightarrow y}$ 。

(1b) (类型变量的重命名)

如果  $\beta$  不在  $M$  中出现, 则  $\lambda \alpha : *.M =_\alpha \lambda \beta : *.M[\alpha := \beta]$ 。

如果  $\beta$  不在  $M$  中出现, 则  $\Pi \alpha : *.M =_\alpha \Pi \beta : *.M[\alpha := \beta]$ 。

(2), (3a), (3b), (3c) (相容性, 自反性, 对称性, 传递性) 不变。

扩展  $\beta$ -规约以满足  $\lambda_2$ :

**定义 6.2** (一步  $\beta$ -规约, 对于  $\Lambda_2$ -项的  $\rightarrow_\beta$ )

(1a) (基础, *Basis*, 一阶)  $(\lambda x : \sigma.M)N \rightarrow_\beta M[x := N]$

(1b) (基础, *Basis*, 二阶)  $(\lambda \alpha : *.M)T \rightarrow_\beta M[\alpha := T]$

(2) (相容性) 不变。

需要注意的是排列定理 (Permutation Lemma) 不再允许对于声明的任意排列, 因为类型变量的声明可能会依赖更早的声明, 但如果排列后依然满足  $\lambda_2$ -上下文, 则性质依然保持。