序

[1] 起初，安得烈、康纳、华特三人，在乌特勒支、斯特拉斯克莱德、诺丁汉之地，同心合意，著书立说。[2] 论到依值类型 λ 演算的奥秘，并 Haskell 的实现，都记在下面。[3] 后来有第七通用设计局的白杨翻出来，又有 Gemini 和 Claude 帮助校对。[4] 白杨说，你们须要记着，凡有智慧的，不可妄言程序设计语言理论，免得为自己在火狱中预备了位置；[5] 只有时时刻刻顾念读者的软弱，如同顾念婴孩，才能不被读者轻看。

[6] 凡行函数式之道的众程序员，论到使用依值类型，心里多有踌躇。[7] 他们彼此议论说："这依值类型，岂不叫查验型别的事变为不可知吗？岂不叫那查验的陷入深渊、永无止境吗？这道甚是艰难，谁能守得住呢？"[8] 然而，这一群人却甚是癫狂，喜爱各样繁杂的仪文。[9] 看哪，在那 Haskell 的境内，有广义代数之像，有函数依赖之规，又有各样关联的族谱与高阶的奥秘。[10] 这一切虚妄的规条，他们都乐意去行；惟独那依值类型，他们却厌弃，如同厌弃大麻风一般，唯恐沾染。

[11] 这一族的人，因不认识依值类型的真意，就跌倒了；这也是那拦阻这道普传的绊脚石。[12] 如今虽有许多美好的器皿和言语，是按着依值类型的法度造的，只是其中的奥秘，向他们是隐藏的，他们便不晓得这器皿是如何运行。[13] 那些指着依值类型所写的书卷，本是文士写给文士看的。[14] 这些话语，对于行函数式的人，甚是生涩。[15] 故此，我写这篇，是要除掉这隔阂，叫你们得以明白。

[16] 我先讲论简单类型 λ 演算，就是那初阶的律法；再讲论依值类型 λ 演算，就是那更美的律法。[17] 从初阶到进阶，所需的变更甚少，你们当留心察看。[18] 我不发明新律法，只将那已有的表明出来，并用 Haskell 的言语以此道造出解释器，好叫你们以此为鉴。[19] 这书不是依值类型编程的入门，也不是完整语言的蓝图，乃是为要除掉你们心中的疑惑，引你们进入这奇妙的领域。

简单类型 λ 演算

[20] 起初，有简单类型 λ 演算，称作$\lambda_{\rightarrow}$。这是最小的律法，各样词项都当显明其类，不可隐藏。[21] 这律法是强正规化的：凡被造的，无论如何运行，终必停息。

论样式的定例

[22] 类型的样式有二：一是基本类型，二是函数类型，就是从甲类到乙类的道路。

$$\begin{aligned} \tau &::= \alpha \\ &\mid \tau \rightarrow \tau' \end{aligned}$$

[23] 词项的样式有四：[24] 第一是指明了归属的；[25] 第二是变量；[26] 第三是应用，就是将甲施于乙；[27] 第四是 λ 抽象，就是造作函数的。

$$\begin{aligned} e &::= e :: \tau \\ &\mid x \\ &\mid e\, e' \\ &\mid \lambda x \rightarrow e \end{aligned}$$

[28] 凡词项被求值，就成为值。值或是中性项，或是 λ 抽象。

$$\begin{aligned} v &::= n \\ &\mid \lambda x \rightarrow v \\ n &::= x \\ &\mid n\, v \end{aligned}$$

论求值

[29] 论到求值，$\lambda_\to$ 既是强正规化的，先求何者，后求何者，都是一样的。 [30] 只是我们要将万物求值到底，连那 λ 里面的，也不可放过。

$$\frac{e \Downarrow v}{e :: \tau \Downarrow v} \qquad \frac{}{x \Downarrow x} \qquad \frac{e \Downarrow v}{\lambda x \to e \Downarrow \lambda x \to v}$$

$$\frac{e \Downarrow \lambda x \to v \quad v[x \mapsto e'] \Downarrow v'}{e\,e' \Downarrow v'} \qquad \frac{e \Downarrow n \quad e' \Downarrow v'}{e\,e' \Downarrow n\,v'}$$

[31] 类型注解，在求值之时，都要废去；变量求值，仍得变量。 [32] 惟有应用一事，当看左边的子项： [33] 若是中性项，就与右边同作中性项，求值便止住了； [34] 若是 λ 抽象，就当行 β 归约。 [35] 这归约的事，常生出新的可归约项来，故此当继续求值，不可止息。 [36] 你们看那 id，就是 $\lambda x \to x$；又看那 const，就是 $\lambda x \to \lambda y \to x$。 [37] 它们求值的样式，记在下面：

$$(\mathtt{id} :: \alpha \to \alpha)\, y \Downarrow y$$

$$(\mathtt{const} :: (\beta \to \beta) \to \alpha \to \beta \to \beta)\ \mathtt{id}\ y \Downarrow \mathtt{id}$$

论类型的律法

[38] 论到类型的律法，大都形如 $\Gamma \vdash e :: \tau$。这意思是说，在 Γ 的光照下，那词项 e 本是属乎 τ 类的。 [39] 那 Γ 乃是类型的册子。

$$\begin{aligned} \Gamma ::= {} & \varepsilon \\ | {} & \Gamma, \alpha :: * \\ | {} & \Gamma, x :: \tau \end{aligned}$$

[40] 凡词项和类型中出现的名，却又不受约束的，都必在册子上有名，不可遗漏。 [41] 譬如那 const，若要显明它属乎 $(\beta \to \beta) \to \alpha \to \beta \to \beta$ 这类，那册子上就当记着：

$$\alpha :: *,\ \beta :: *,\ \mathtt{const} :: (\beta \to \beta) \to \alpha \to \beta \to \beta$$

[42] 你们当晓得，那 α 和 β 必须先记在册子上，在 const 前面。

[43] 论到册子的定例，都在这里显明了。

$$\frac{}{\text{valid}(\varepsilon)} \qquad \frac{\text{valid}(\Gamma)}{\text{valid}(\Gamma, \alpha :: *)} \qquad \frac{\text{valid}(\Gamma) \quad \Gamma \vdash \tau :: *}{\text{valid}(\Gamma, x :: \tau)}$$

$$\frac{\Gamma(\alpha) = *}{\Gamma \vdash \alpha :: *}\ [\text{TVAR}] \qquad \frac{\Gamma \vdash \tau :: * \quad \Gamma \vdash \tau' :: *}{\Gamma \vdash \tau \to \tau' :: *}\ [\text{FUN}]$$

[44] 末后的两条诫命，是论到类型的良构：凡类型中一切的名，若是不受约束的，就必在册子上有名，这类型才是全备的。 [45] 在良构性的律法，并后续的类型律法中，凡提到这册子的，我们都以为它是实在有效的。

[46] 你们当谨守，$\lambda_\to$ 并非多态，各类型的名都指着实实在在的类型，不可随意更改。

[47] 如今我将类型的律法陈明。 [48] 我们不推断那被 λ 遮盖之变量的类型，只作查验。 [49] 然而，凡带了印记的词项、变量、并应用，我们都能断言其类。 [50] 凡查验所须知的类型，都注明了 $::_\downarrow$；凡查验所得的类型，都注明了 $::_\uparrow$。 [51] 这不过是叫你们心里明白，到了行出来的时候，两样的分别就显明了。

$$\frac{\Gamma \vdash \tau :: * \quad \Gamma \vdash e ::_{\downarrow} \tau}{\Gamma \vdash (e :: \tau) ::_{\uparrow} \tau}\ [\text{ANN}] \qquad \frac{\Gamma(x) = \tau}{\Gamma \vdash x ::_{\downarrow} \tau}\ [\text{VAR}]$$

$$\frac{\Gamma \vdash e ::_{\uparrow} \tau \to \tau' \quad \Gamma \vdash e' ::_{\downarrow} \tau}{\Gamma \vdash e\, e' ::_{\downarrow} \tau'}\ [\text{APP}]$$

$$\frac{\Gamma \vdash e ::_{\uparrow} \tau}{\Gamma \vdash e ::_{\downarrow} \tau}\ [\text{CHK}] \qquad \frac{\Gamma, x :: \tau \vdash e ::_{\downarrow} \tau'}{\Gamma \vdash \lambda x \to e ::_{\downarrow} \tau \to \tau'}\ [\text{LAM}]$$

[52] 凡带了印记的，就按着印记断定它。凡是变量，就当在册子上查考。 [53] 论到应用，必先察看那左边的——它必有函数的样式；然后将那右边的与函数的输入类型相比对，若相合，就以函数的返回类型为果效。 [54] 末后的两条，是查验的律法。若推断出的与所定的相合，这词项就算为义了。 [55] 只有 λ 抽象，须能被查验为函数类型。我们当在扩展的册子中，查验那 λ 的身体。 [56] 虽然那 [CHK] 似乎能包罗万象，但在律法中，并没有推断 λ 类型的条文，也没有查验印记、变量或应用的条文。 [57] 故此，这律法极容易行出来，如同行在平原。 [58] `id` 和 `const` 的类型判断，是按着这律法得出的，都记在下面：

$$\alpha :: *,\ y :: \alpha \vdash (\mathtt{id} :: \alpha \to \alpha)\, y :: \alpha$$
$$\alpha :: *,\ y :: \alpha,\ \beta :: * \vdash (\mathtt{const} :: (\beta \to \beta) \to \alpha \to \beta \to \beta)\ \mathtt{id}\ y :: \beta \to \beta$$

依值类型 λ 演算

[59] 时候到了，我们要离弃那初阶的道理，竭力进到完全的地步，就是依值类型 λ 演算，称作 λ_{Π}。 [60] 在这章的起头，我要先讲论这更改中的两个精意。 [61] 随后，我要陈明抽象语法、求值和类型的定例，凡与$\lambda_{\to}$不同的，都必指明。

论依值函数空间

[62] 在 Haskell 的全地，凡要立多态函数的，像那恒等函数一般，原是可以的：

```
id :: ∀α. α -> α
id = \x -> x
```

[63] 因有多态的恩典，人便不必为各样的物，或是整数，或是真伪，重修一样的祭坛，免得劳烦。 [64] 若视多态为类型的抽象，这理就显明了。 [65] 如此，恒等函数便受了两个参数：一是类型 α，一是属 α 的值。 [66] 凡求告这名的，须指明类型的名，好叫它成形：

```
id :: ∀α. α -> α
id = \(α :: *) (x :: α) -> x

id Bool True :: Bool
id Int 3 :: Int
```

[67] 多态既许了类型上的抽象，我们何必另求别的呢？ [68] 你们看那数据的样式：

```
data Vec0 α = Vec0
data Vec1 α = Vec1 α
data Vec2 α = Vec2 α α
data Vec3 α = Vec3 α α α
```

[69] 这些样式虽多，法度却是一样。我们愿有一个单一的族，按着数目的多寡分列：

$$\forall \alpha :: *. \forall n :: \mathtt{Nat}. \mathtt{Vec}\ \alpha\ n$$

[70] 只是在 Haskell 地，这事却行不通，，因为这 `Vec` 是对值 n 抽象的。

[71] 论到依值函数的度量，其记号乃是 $\forall$，这原比寻常函数空间 $\rightarrow$ 更大；因它准许函数返回值的类型，倚赖输入的值。[72] 那参数的多态，在 Haskell 中原是闻名的，实乃依值函数的一类。故此，我们取了这符号 $\forall$，作我们中间的记号。[73] 只是多态的权柄，仅及于类型；依值函数空间却不如此，其权柄所及的，不独是类型而已。[74] 看哪，先前所提的 Vec，那确是合宜的依值类型。[75] 譬如，我们可以为那用在 `Vec` 上的恒等函数 `id`，注上这样的类型：

$$\forall \alpha :: * \,.\, \forall n :: \texttt{Nat} \,.\, \forall v :: \texttt{Vec}\ \alpha\ n \,.\, \texttt{Vec}\ \alpha\ n$$

[76] 你们看，那类型 v 未曾在返回类型中显现：这就是非依值函数空间，是行 Haskell 的人所熟知的。[77] 与其立那无用的 v，不如为非依值的情况用常规的箭头。[78] 所以上面的类型注解也可以写成：

$$\forall \alpha :: * \,.\, \forall n :: \texttt{Nat} \,.\, \texttt{Vec}\ \alpha\ n \rightarrow \texttt{Vec}\ \alpha\ n$$

[79] Haskell 的工匠虽用私意模拟依值类型空间，就如在类型层面上造出自然数。[80] 然而，类型层面的自然数，与值层面的自然数，中间有深渊隔定，不能过去。[81] 工匠便不得不在两边，重修许多的道理。[82] 虽行奇事，能将值层面的物升到类型层面，但要用这些类型作计算，总要费许多的力气。[83] 惟独依值类型，我们可以直接用值将类型参数化，并且仍守常规的求值定例——这定例我们快要看见了。

论一切皆词项

[84] 值若得以进入类型的圣所，那词项、类型、与种类中间隔断的墙，就拆毁了。[85] 并不分词项、类型、种类，都在依值里成为一了。[86] 从前在 Haskell 地，有记号名唤 “::”，将万象各从其类，归于不同的层级：[87] 论到 $0 :: \texttt{Nat}$，那 0 原是值，`Nat` 原是类型；[88] 论到 $\texttt{Nat} :: *$，那 `Nat` 又是类型，$*$ 乃是种类。[89] 看哪，如今这一切，都同归于一，成为词项了。[90] 那 $0 :: \texttt{Nat}$ 与 $\texttt{Nat} :: *$ 的理虽未废去，只是那 “::”，现今乃是连络词项与词项的。[91] 我们虽按着外貌称 $\rho :: *$ 中的 ρ 为类型，亦称那 $*$ 为种类，然而在灵里，都是一样的词项。[92] 故此，凡律法上的规条，无论在何处，皆可运用；[93] 或在天上，或在地下，也可行抽象应用之事。[94] 如今这依值类型的奥秘，我们已经晓得了。

论样式的定例

[95] 如今我们不再将词项、类型、种类分别为圣，乃是将万有都归于一，统称为词项。[96] 凡在各层各界的，都当归入这唯一的律法中，就记在下面：

$$\begin{aligned} e, \rho, \kappa ::= {} & e :: \rho \\ \mid {} & * \\ \mid {} & \forall x :: \rho \,.\, \rho' \\ \mid {} & x \\ \mid {} & e\, e' \\ \mid {} & \lambda x \rightarrow e \end{aligned}$$

[97] 凡有高亮之处，是与前约不同。[98] 如今我们也用 ρ 和 κ 指涉那作类型和种类的词项。[99] 原本在类型和种类的律法中所有的，如今也都归入这新约里了。[100] 看哪，种类 $*$ 如今也是一个词项；依值函数空间涵盖了原本的箭头种类和箭头类型。[101] 类型变量和词项变量，如今也合而为一了。

论求值的定例

[102] 看哪，这新立的求值定例，记在下面。 [103] 除了那新添的两样，其余的都与$\lambda_{\rightarrow}$无异。 [104] 你们不要惊奇，说：“求值的律法，怎竟临到类型身上了呢？” [105] 这原是好的，因为依值类型的权柄，就在于将值与类型调和，叫我们在类型之中，也能行计算的事。

$$\frac{}{* \Downarrow *}$$

$$\frac{\rho \Downarrow \tau \quad \rho' \Downarrow \tau'}{\forall x :: \rho\,.\,\rho' \Downarrow \forall x :: \tau\,.\,\tau'}$$

[106] 论到那 $*$，求值之后仍是自己。 [107] 论到依值函数的空间，当递归地察看其中的输入与输出。 [108] 值的样式，如今更增添了：

$$\begin{aligned} v, \tau ::= {} & n \\ \mid {} & * \\ \mid {} & \forall x :: \tau\,.\,\tau' \\ \mid {} & \lambda x \rightarrow v \end{aligned}$$

[109] 如今我们用 τ 这记号，指涉那作为类型的值。

论类型的律法

[110] 在未曾观看律法之先，当思想那册子。 [111] 如今万物皆是词项，册子的样式与定例，便都简化了。

$$\begin{aligned} \Gamma ::= {} & \varepsilon \\ \mid {} & \Gamma, x :: \tau \end{aligned}$$

$$\frac{}{\text{valid}(\varepsilon)} \qquad \frac{\text{valid}(\Gamma) \quad \Gamma \vdash \tau ::_{\downarrow} *}{\text{valid}(\Gamma, x :: \tau)}$$

[112] 这册子上，如今只有一样条目，就是变量与其类型。 [113] 凡记在其中的，都是求值过的。 [114] 那查验册子的定例，不再指涉类型良构的判断，乃是指涉我们将要立的律法。 [115] 这一条，是为要确立 τ 之中，不含未知的变量。

[116] 类型的律法，记在下面。 [117] 如今这律法，是与上下文、词项、并那值相关的。 [118] 凡是类型，都当尽早求值。 [119] 当推断的，仍是用 $::_{\uparrow}$；当查验的，仍是用 $::_{\downarrow}$。 [120] 那 $*$ 与 $\forall$，都是我们可以推断的。

$$\frac{\Gamma \vdash \rho ::_{\downarrow} * \quad \rho \Downarrow \tau \quad \Gamma \vdash e ::_{\downarrow} \tau}{\Gamma \vdash (e :: \rho) ::_{\uparrow} \tau} \quad [\text{ANN}]$$

$$\frac{}{\Gamma \vdash * ::_{\uparrow} *} \quad [\text{STAR}] \qquad \frac{\Gamma \vdash \rho ::_{\downarrow} * \quad \rho \Downarrow \tau \quad \Gamma, x :: \tau \vdash \rho' ::_{\downarrow} *}{\Gamma \vdash \forall x :: \rho\,.\,\rho' ::_{\uparrow} *} \quad [\text{PI}]$$

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x ::_{\downarrow} \tau} \quad [\text{VAR}] \qquad \frac{\Gamma \vdash e ::_{\uparrow} \forall x :: \tau\,.\,\tau' \quad \Gamma \vdash e' ::_{\downarrow} \tau \quad \tau'[x \mapsto e'] \Downarrow \tau''}{\Gamma \vdash e\, e' ::_{\downarrow} \tau''} \quad [\text{APP}]$$

$$\frac{\Gamma \vdash e ::_{\uparrow} \tau}{\Gamma \vdash e ::_{\downarrow} \tau} \quad [\text{CHK}] \qquad \frac{\Gamma, x :: \tau \vdash e ::_{\downarrow} \tau'}{\Gamma \vdash \lambda x \rightarrow e ::_{\downarrow} \forall x :: \tau\,.\,\tau'} \quad [\text{LAM}]$$

[121] 论到那受了印记的 [ANN]，其定例有二更改： [122] 其一，那印记 ρ，不再察验其全备与否，乃要察验其归属； [123] 其二，因 ρ 未必是值，当先求值，方能使用。

[124] 论到 $*$ [STAR]，其类型仍是 $*$。 [125] 虽有闲杂人议论这事，只是为了成全这律法的纯全，我们便不听从。

[126] 论到依值函数空间 [PI]，这原与从前那箭头类型的定例 [FUN] 相仿。 [127] 那输入的 ρ 与输出的 ρ'，都当归于 $*$ 之下。 [128] 只是 ρ' 之中，如今既有了 x 的影儿，察验之时，就当将 x 记在册子上，不可遗漏。

[129] 论到应用 [APP]，那函数必有依值函数的类型。 [130] 结果的类型中，当将那形参 x 涂抹，换作实参 e'。

[131] 论到察验 [CHK]，仍与从前一样。 [132] 只是如今所比对的，乃是求值过的类型。 [133] 若不求值，那 $\texttt{Vec}\ \alpha\ 2$ 与 $\texttt{Vec}\ \alpha\ (1+1)$ 岂不就分别为二了吗？ [134] 如今按着这律法，它们原是合一的。

[135] 外邦人虽有转换的律法，只是那律法不是凭着语法引导的。 [136] 惟独在察验注解的类型时，我们方才使用。

[137] 末了，论到 Lambda 抽象 [LAM]。 [138] 其类型如今更变了，乃是依值函数类型。 [139] 那被约束的 x，不独在函数体 e 中显现，也可在返回类型 τ' 中显现。 [140] 故此察验之时，当用那扩展过的册子。 [141] 这些事的总意就是： [142] 函数空间泛化为依值函数空间；类型与种类，皆是词项。