简单类型 λ 演算

简单类型 λ 演算是一种非常简单的类型系统。在本文中，我们将从简单类型 λ 演算的基本性质出发，介绍四个不那么平凡的结果：

- **强正则定理**：简单类型 λ 演算中，不存在以有类型的项开始的无限长的 β 规约序列；
- **类型推导算法：** 对于 Curry 风格（变量不注明类型）的 λ 演算，存在高效的算法求出 λ 项的“主类型”；
- **组合子逻辑**：一个仅用 2 个组合子和组合子应用形成的演算系统，不需要任何变量；
- **Curry-Howard 同构**：简单类型 λ 演算与直觉主义命题逻辑的自然演绎系统对应，简单类型组合子逻辑与直觉主义希尔伯特公理系统对应。

本文的目录如下。

类型系统

简单类型 λ 演算（Simply Typed λ-Calculus），或称 $\lambda_{\to}$，在无类型 λ 演算的基础上引入了一个基本的类型系统。在简单类型 λ 演算中，类型要么是基本类型 $T \in \mathcal{T}$，要么形如 $\varphi \to \psi$，非形式化地，表示参数为 φ、返回值为 ψ 的函数。其中箭头是右结合的，即 $\varphi \to \psi \to \rho := \varphi \to (\psi \to \rho)$。

在简单类型 λ 演算中，λ 抽象中的变量需要标记类型，即

$$\lambda x : \varphi .\, e$$

根据变量的类型，我们可以根据一组类型规则确定一个 λ 项的类型，即

$$\frac{}{?, x : \varphi \vdash x : \varphi} \qquad \frac{?, x : \varphi \vdash e : \psi}{? \vdash (\lambda x : \varphi .\, e) : \varphi \to \psi} \qquad \frac{? \vdash e_1 : \varphi \to \psi \quad ? \vdash e_2 : \varphi}{? \vdash (e_1\ e_2) : \psi}$$

其中，上下文 ? 的定义扩展为变量和类型的序对 $x : \varphi$ 构成的集合。$\mathrm{dom}\ ?$ 表示上下文中所有变量构成的集合，$\mathrm{range}\ ?$ 表示所有类型构成的集合。其中，$?, x : \varphi$ 表示将 ? 中加入 $x : \varphi$（如果 $x \in \mathrm{dom}\ ?$，覆盖已有的类型）。

类型系统排除掉了一些“非正则”的 λ 项。举例而言，$\lambda x .\, x\ x$ 是无类型 λ 演算中的一个项，但在无论给 x 任何的类型，它都不是合法的 λ 项。事实上，我们将会证明简单类型 λ 演算的所有项都是**强正则（Strongly Normalized）**的，即不存在一个从有类型的 λ 开始的无限长的 β 规约序列。

Curry 风格的 $\lambda_{\to}$

上面所介绍的简单类型 λ 演算被称为 **Church 风格的（à la Church）**，而另一种类似的系统称为 **Curry 风格的（à la Curry）**。Curry 风格的 $\lambda_{\to}$ 并不在变量上标注其类型，相反，如果一个 λ 项可以为每个变量分配类型，使其在 Church 风格 $\lambda_{\to}$ 中具有类型，就称其是可类型化的（Typable）。一个可类型化的 λ 项可能具有多种合法的类型分配，例如 $\lambda x .\, x$ 有多种可能的类型

$$\mathrm{bool} \to \mathrm{bool}, \mathrm{int} \to \mathrm{int}, (\mathrm{int} \to \mathrm{int}) \to (\mathrm{int} \to \mathrm{int}), \ldots$$

类型的基本性质

性质 1（Generation Lemma） 类型规则可以反向使用，即

1. $? \vdash x : \varphi$，那么 $x : \varphi \in ?$
2. $? \vdash (e_1\ e_2) : \psi$，那么存在 φ 使得 $? \vdash e_1 : \varphi \to \psi$ 且 $? \vdash e_2 : \varphi$
3. $? \vdash (\lambda x : \varphi .\, e) : \varphi \to \psi$，那么 $?, x : \varphi \vdash e : \psi$

性质 2（Substitution Lemma） 类型不会由于类型名称/项替换而改变

1. $? \vdash e : \varphi$，那么 $?[\sigma \mapsto \rho] \vdash e : \varphi[\sigma \mapsto \rho]$
2. $?, x : \sigma \vdash e : \varphi$，并且 $? \vdash e' : \sigma$，那么 $? \vdash e[x \mapsto e']$

- 证明思路：施归纳于 $?, x : \sigma \vdash e : \varphi$ 确定类型的结构。

性质 3（β-Reduction Lemma） β-reduction 不会改变项的类型

- 证明思路：施归纳于 β-reduction 的结构，利用替换引理。

定理（Church-Rosser Theorem）： 对于任意的 $e \twoheadrightarrow_\beta e_1, e \twoheadrightarrow_\beta e_2$，存在一个 e_3，使得 $e_1 \twoheadrightarrow_\beta e_3$，且 $e_2 \twoheadrightarrow_\beta e_3$

- 证明思路：由无符号 λ 演算的 Church-Rosser 定理，及 β 规约引理证明。

类型对应的树

为了描述的方便，我们可以将类型和树相互对应。我们称 T 是类型 φ 对应的树，如果

1. φ 是基本类型，T 仅包含一个标有 φ 的结点；
2. $\varphi = \sigma_1 \to \sigma_2 \to \cdots \to \sigma_k$，$T$ 的根节点标有 φ，且 k 个孩子 $T_1, T_2, \ldots, T_k$ 分别是 $\sigma_1, \sigma_2, \ldots, \sigma_k$ 对应的树。

类型对应的树并不一定是唯一的。以类型 $\alpha \to (\alpha \to \alpha) \to \alpha$ 为例，

$$\frac{\alpha \to (\alpha \to \alpha) \to \alpha}{\alpha \qquad \frac{(\alpha\to\alpha)\to\alpha}{\frac{\alpha\to\alpha}{\alpha \quad \alpha} \qquad \alpha}} \qquad\qquad \frac{\alpha \to (\alpha \to \alpha) \to \alpha}{\alpha \qquad \frac{\alpha\to\alpha}{\alpha \quad \alpha} \qquad \alpha}$$

都是合法的类型对应的树。其中，我们关心两个特殊的树，即

1. 每个非基本类型均有两个儿子，这棵树称为类型**对应的二叉树**；
2. 对于每个结点 T，其最后一个孩子 T_k 对应于基本类型，这棵树称为类型**对应的多叉树。**

二叉树和多叉树表示体现了我们看待类型的两种视角，分别对应了一个函数的 curry/uncurry 化。其中，多叉树表示将一个非基本类型的 λ 项看作一个多元函数，每个子结点对应于每一项参数的类型。

定义（类型的高度）： 定义类型的高度为对应二叉树的高度。换言之

1. $h(\alpha) = 1$
2. $h(\varphi \to \psi) = \max\{h(\varphi), h(\psi)\} + 1$

$\lambda_\to$ 的表达能力

自然数

设 α 是任意类型，类似无类型 λ 演算中 Church 数的定义，我们有

$$\begin{aligned}
0 &:= \lambda f : \alpha \to \alpha.\, \lambda x : \alpha.\, x \\
1 &:= \lambda f : \alpha \to \alpha.\, \lambda x : \alpha.\, f\ x \\
2 &:= \lambda f : \alpha \to \alpha.\, \lambda x : \alpha.\, f\ (f\ x) \\
\vdots &:= \vdots
\end{aligned}$$

根据类型规则，我们有 $int := (\alpha \to \alpha) \to (\alpha \to \alpha)$。容易验证自然数的基本运算可以在 $\lambda_{\to}$ 中定义。

布尔值

设 α 是任意类型，我们可以定义布尔值 $bool_\alpha := \alpha \to \alpha \to \alpha$，其中

$$\begin{aligned} true_\alpha &:= \lambda x : \alpha.\, \lambda y : \alpha.\, x \\ false_\alpha &:= \lambda x : \alpha.\, \lambda y : \alpha.\, y \end{aligned}$$

值得注意的是，$true$ 和 $false$ 并不能像原先一样充当一般的 if-then-else 语句。事实上，使用 $true_\alpha$ 实现的 if-then-else 语句要求 then 和 else 中的值必须具有**相同的类型**，且均为 α。举例而言，

$$\begin{aligned} true_{int}\ 1\ 2 &: int \\ false_{int}\ 1\ 2 &: int \\ true_{int}\ 1\ (x : \alpha) &: untypeable \\ true_{int}\ (x : \alpha)\ (y : \alpha) &: untypeable \end{aligned}$$

为了语言使用的方便，可以新增基本类型 bool 和 ITE 语句，并定义类型规则

$$\frac{?\vdash e_1 : bool \quad ?\vdash e_2 : \varphi \quad ?\vdash e_3 : \varphi}{?\vdash \mathrm{ITE}(e_1, e_2, e_3) : \varphi}$$

以及各种逻辑连接词，例如 $\mathrm{and}, \mathrm{or}, \mathrm{not}$ 等等。

正则定理

正则和值

定义（正则和值）：

1. $\lambda x : \varphi, e$ 和 $x\ e_1\ e_2\ \ \ldots e_k$ 称为**值（Value）**
2. 定义 e 为**正则的（Normalized）**，如果不存在 e' 使得 $e \to_\beta e'$
3. 称 e 是**可正则化（Normalizable）**的，如果其存在一个正则表示。称 e' 是 e 的**正则表示（Normal Form）**，如果 e' 是正则的，且 $e \twoheadrightarrow_\beta e'$，记作 $e \Downarrow e'$

非形式化的，不能继续计算（即在顶层执行 λ 应用）的项称为值，不能继续规约的项是正则的。

性质 1：正则的项是值

证明：施归纳于 λ 项的结构。如果一个正则的项具有 $e_1\ e_2$ 的形态，由于 $e_1\ e_2$ 是正则的，e_1 也是正则的，从而 e_1 是值。那么

1. 如果 e_1 形如 $\lambda x : \varphi, e'$，那么 $e_1\ e_2$ 可以进行一次 β 规约，这与 λ 是正则的矛盾。

2. 如果 e_1 形如 $x\ e'_1\ e'_2\ \ldots\ e'_k$，那么 $e_1\ e_2$ 为

$$e_1\ e_2 = x\ e'_1\ e'_2\ \ldots\ e'_k\ e_2$$

从而也是值。

性质 2： 如果 e 是可正则化的，那么其正则表示唯一

证明： 若存在两个正则表示，根据 Church-Rosser 定理，其正则表示必然唯一。

我们将要证明两个正则定理，**弱正则定理（Weak Normalization Theorem）** 和 **强正则定理（Strong Normalization Theorem）**。其中，弱正则定理是强正则定理的平凡推论，为了思维的完整性，我们首先给出其证明。

弱正则定理

定理（弱正则定理，Weak Normalization Theorem）： 在 $\lambda_{\to}$ 中，任何有类型的项都是可正则化的。更严格地，如果 $?\vdash e : \varphi$，那么 e 是可正则化的。

证明（Turing，Prawitz）： 不妨称 $(\lambda x : \varphi.\, e_1)\ e_2$ 为一个可规约项，定义其高度为 $\lambda x : \varphi.\, e_1$ 类型的高度，定义 λ 项 e 的高度 $h(e)$ 为其中高度最大的可规约项的高度，$n(e)$ 为 e 中高度为 $h(e)$ 的 λ 应用对数。施归纳于 $(h(e), n(e))$ 的字典序，证明所有有类型的项都是可正则化的。

1. 当 $h(e) = 1, n(e) = 0$ 时，结论是显然的。
2. 取 e 中最靠右的、高度为 $h(e)$ 的可规约项 $\Delta = (\lambda x : \varphi.\, e_1)\ e_2$。由于 Δ 的选取，$h(e_1) < h(e)$，且 $h(e_2) < h(e)$。让我们证明 $e_1[x \mapsto e_2]$ 的高度小于 $h(e)$，从而可以说明使用 $e_1[x \mapsto e_2]$ 代替 Δ 得到的 λ 项 e'，$(h(e'), n(e'))$ 具有较小的字典序。

   假设 $e_1[x \mapsto e_2]$ 中存在一个可规约项 Δ' 满足高度大于等于 $h(e)$，那么

   a. 可规约项不包含由 $[x \mapsto e_2]$ 产生的项，其也存在于 e_1 中。由于 $h(e_1) < h(e)$，这是不可能的；
   b. 可规约项以 $[x \mapsto e_2]$ 产生的项作为第一项，即形如 $\Delta' = e_2\ o$，这是不可能的，因为 $h(e_2) < h(e)$；
   c. 可规约项以 $[x \mapsto e_2]$ 产生的项作为第二项，即形如 $\Delta' = o\ e_2$，那么 $o\ x$ 构成了一个 e_1 中的相同高度的可规约项，由于 $h(e_1) < h(e)$，这时不可能的。
   d. 可规约项以 $[x \mapsto e_2]$ 产生的项作为第一项和第二项，这是不可能的，因为 $x\ x$ 不可能存在于有类型的项中，对于任意 $x : \varphi$。
   e. 可规约项在 e_2 内部，由于 $h(e_2) < h(e)$，这是不可能的。

弱正则定理说明，如果我们不断地选择最右侧、最高的可规约项进行规约，任何有类型的 λ 项总能在有限步内化为正则项。根据 Church-Rosser 定理，任何两个 $\beta-$等价的 λ 项都具有相同的正则表示，因此这提供了一个判定两个 λ 项 $\beta-$等价性的算法。

弱正则定理的不足之处在于，它并没有说明任何有类型的项在“简明”的求值规则下可以在有限步终止。一般来说，常见“简明”的规约规则可以分为 Call by Value 和 Call by Name 两种方式。Call by Value 首先将 λ 应用的参数规约，然后做一步 β 规约；Call by Name 则直接进行 β 规约。举例而言，Call by Value 的 λ 演算求值规则可以定义如下

$$
\begin{array}{ll}
\text{Value-Rule} & \dfrac{}{\text{Val}(\lambda x:\varphi.\,e)} \\
\text{Function-Reduce} & \dfrac{e_1 \Rightarrow e_1'}{e_1\ e_2 \Rightarrow e_1' e_2} \\
\text{Argument-Reduce} & \dfrac{\text{Val}(e_1) \quad e_2 \Rightarrow e_2'}{e_1\ e_2 \Rightarrow e_1\ e_2'} \\
\text{Beta-Reduction} & \dfrac{\text{Val}(e_2)}{(\lambda x:\varphi.\,e_1)\ e_2 \Rightarrow e_1[x \mapsto e_2]}
\end{array}
$$

容易说明，如果 $\vdash e$ 且 e 不是值，那么有且仅有一条求值规则可以执行，即求值过程不会“卡住”。我们希望说明，任何有类型的 λ 项的求值过程可以在有限步内终止。事实上，无论以何种顺序，任何有类型的 λ 项进行求值的过程总是终止的。

强正则定理

定理（强正则定理，Strong Normalization Theorem）： 不存在以有类型的 λ 项开始的无限长的 β 规约序列 [1]。

证明： 首先归纳地定义 SN e 表示任给 $e_1, e_2, \ldots, e_k$，$k \geq 0$，满足

1. 对于 $1 \leq i \leq k$，恒有 SN e_i
2. $e\ e_1\ e_2\ \ldots\ e_k$ 是有类型的

那么 $e\ e_1\ e_2\ \ldots\ e_k$ 不存在无限长的 β 规约序列。由于 e_i 类型的高度小于 e 类型的高度，SN 是良定义的。

定义 SN* e，如果对于任意的 $\{y_1, y_2, \ldots, y_k \mid \text{SN}\ y_i\}$，都有SN $e[\forall i, x_i \mapsto y_i]$，其中 $\{x_1, x_2, \ldots, x_k\} = \text{FV}\ e$。为了简便，将 $e[\forall i, x_i \mapsto y_i]$ 记作 $\gamma(e)$。

增强定理： 对于任意的 $? \vdash e : \varphi$，都有 SN* e

显然增强定理确实是原定理的增强。为证明增强定理，我们给出一个引理。

引理： 对于任意 $? \vdash e : \varphi$ 和 $e \rightarrow_\beta e'$，那么 SN $e \Rightarrow$ SN e'

引理的证明： 由于 β 规约序列

$$e\ e_1\ \ldots\ e_k \rightarrow_\beta e'\ e_1\ \ldots\ e_k \twoheadrightarrow_\beta \ldots$$

始终是有穷的，这一结论显然。

增强定理的证明： 施归纳于 $? \vdash e : \varphi$ 的步骤，证明 $\mathrm{SN}^\star\ e$。

1. 如果 $e = x$，对于任何合法的 γ，$\mathrm{SN}\ \gamma(e)$ 成立。
2. 如果 $e = o_1\ o_2$，只需证明 $\mathrm{SN}(\gamma(o_1\ o_2))$，这等价于 $\mathrm{SN}((\gamma(o_1)\ \gamma(o_2)))$。根据归纳假设，$\mathrm{SN}^\star(o_1)$ 且 $\mathrm{SN}^\star(o_2)$，那么也有 $\mathrm{SN}(\gamma(o_1))$ 和 $\mathrm{SN}(\gamma(o_2))$。任给 $e_1, e_2, \ldots, e_k$ 满足 $\mathrm{SN}\ e_i$，由于
    a. $\mathrm{SN}(\gamma(o_1))$
    b. $\mathrm{SN}(\gamma(o_2)), \mathrm{SN}(e_1), \ldots, \mathrm{SN}(e_k)$

    根据 $\mathrm{SN}(\gamma(o_1))$ 的定义，可知 $\gamma(o_1)\ \gamma(o_2)\ e_1\ \ldots\ e_k$ 不存在无限长的 β 规约序列。
3. 如果 $e = \lambda x : \varphi.\, o$，只需证明 $\mathrm{SN}(\gamma(\lambda x : \varphi.\, o))$，这等价于 $\mathrm{SN}(\lambda x : \varphi.\, \gamma(o)))$。下面用反证法。如果有 $e_1, e_2, \ldots, e_k$ 满足 $\mathrm{SN}\ e_i$，使得 $(\lambda x : \varphi.\, \gamma(o)))\ e_1\ \ldots\ e_k$ 存在一个无限长的 β 规约序列，其一定形如

$$
\begin{aligned}
&(\lambda x : \varphi.\, \gamma(o)))\ e_1\ \ldots\ e_k \\
\twoheadrightarrow_\beta\ &(\lambda x : \varphi.\, \gamma(o)))\ e_1'\ \ldots\ e_k' \\
\rightarrow_\beta\ &\gamma(o)[x \mapsto e_1']\ e_2'\ \ldots\ e_k' \\
\twoheadrightarrow_\beta\ &\ldots
\end{aligned}
$$

    根据引理 1，对于 $1 \le i \le k$，均有 $\mathrm{SN}\ e_i'$。又因为可以构造合法的 γ'，满足 $\gamma'(o) = \gamma(o)[x \mapsto e_1']$，从而根据归纳假设 $\mathrm{SN}^\star(o)$，有 $\mathrm{SN}(\gamma'(o))$。那么

$$
\gamma(o)[x \mapsto e_1']\ e_2'\ \ldots\ e_k' = \gamma'(o)\ e_2'\ \ldots\ e_k'
$$

    不存在无限长的 β 规约序列，这与假设矛盾。

综上所述，增强定理成立，因而原定理也成立。

类型推导

主类型

对于一个 Curry 风格的 λ 项，其可能具有多种不同的合法类型指派，且不同的类型指派之间有着有趣的关系。以 $\lambda x.\, \lambda y.\, x\ y$ 为例，类型指派 $\{(x : \alpha \to \beta), (y : \alpha)\}$ 对应的项的类型为 $(\alpha \to \beta) \to \alpha \to \beta$，而另一个类型指派 $\{(x : \alpha \to \alpha), (y : \alpha)\}$ 对应的类型为 $(\alpha \to \alpha) \to \alpha \to \alpha$。如果将 α, β 都看作待取值的“变量”，前者是一个更一般的类型指派，对应的项的类型也更通用。非形式化地，我们将一个 λ 项“最通用”的类型称为其**主类型 (Principal Type)**。

为了描述主类型，我们需要定义**类型变量**。类型变量可以被任意替换成其他类型，表示在任意替换后得到的类型仍是原先 λ 项的类型。举例而言，如果将 α, β 看作基本类型，$(\alpha \to \beta) \to \alpha \to \beta$ 是 λ 项 $\lambda x.\, \lambda y.\, x\ y$ 的主类型，将 α 替换为 β 即可得到主类型的一个特例。

我们定义类型的替换 S 是一个替换序对 $\alpha \mapsto \varphi$ 的集合，表示将类型变量 α 替换为类型 φ。其中 dom 和 range 的定义同 λ 项的替换。替换作用于类型的规则可以简单地定义为

$$\begin{aligned} S(\alpha) &:= \alpha, \alpha \notin \mathrm{dom}\ S; \\ S(\alpha) &:= \varphi, \alpha \mapsto \varphi \in S; \\ S(\varphi \to \psi) &:= S(\varphi) \to S(\psi). \end{aligned}$$

对于两个类型 φ, ψ，如果存在一个 S 使得 $S(\varphi) = \psi$，我们称 φ 比 ψ 更通用，记作 $\varphi \leq_S \psi$。显而易见，由于替换是可以合成的，$\leq_S$ 构成一个偏序关系。定义 $\varphi \sim_S \psi$ 如果 $\varphi \leq_S \psi$ 且 $\psi \leq_S \varphi$。

根据这一偏序关系，我们可以定义出下界、下确界以及上界、上确界。接下来我们证明任意两个元素均有上、下确界，从而类型在 $\leq_S$ 关系下构成一个偏序格。

引理：类型在 $\leq_S$ 关系下构成一个偏序格。

证明：为此，仅需证明任意两个元素均有上下确界。由于两个方向证明类似，我们只证明下确界。对于任意的类型 $\varphi, \psi, \varphi', \psi'$ 和类型变量 α，我们定义 $\wedge$ 运算为

$$\begin{aligned} \alpha \wedge \varphi &:= \alpha; \\ \varphi \wedge \alpha &:= \alpha; \\ \varphi \to \psi \wedge \varphi' \to \psi' &:= (\varphi \wedge \psi) \to (\varphi' \wedge \psi'). \end{aligned}$$

用归纳法容易说明 $\varphi \wedge \psi$ 是 φ 和 ψ 的下界，下面证明 $\varphi \wedge \psi$ 正是 φ, ψ 的下确界。施归纳于 $\wedge$ 的计算，证明对于任意的 ρ 满足 $\rho \leq_S \varphi$ 且 $\rho \leq_S \psi$，均有 $\varphi \wedge \psi \leq_S \rho$。讨论

1. φ 或 ψ 是类型变量，由于 $\varphi \wedge \psi$ 一定是类型变量，我们有 $\varphi \wedge \psi \leq_S \rho$；
2. 若 $\varphi = \varphi_1 \to \varphi_2$ 且 $\psi = \psi_1 \to \psi_2$，由于 $\rho \leq_S \varphi$ 有 $\rho = \rho_1 \to \rho_2$ 且 $\rho_i \leq_S \varphi_i$，同理有 $\rho_i \leq_S \psi_i$。根据归纳假设，我们有 $\varphi_i \wedge \psi_i \leq_S \rho_i$。根据替换的定义，有 $\varphi \wedge \psi \leq_S \rho$。

类似地，上确界的运算 $\vee$ 为

$$\begin{aligned} \alpha \vee \varphi &:= \varphi; \\ \varphi \vee \alpha &:= \varphi; \\ \varphi \to \psi \vee \varphi' \to \psi' &:= (\varphi \vee \psi) \to (\varphi' \vee \psi'). \end{aligned}$$

这一引理保证了“最通用”这一说法是良定义的。对于一个项不同的类型 $\varphi_1, \ldots, \varphi_k$，根据格的性质，我们能够找到它们的下界 $\varphi = \varphi_1 \wedge \cdots \wedge \varphi_k$，使得任何一个合法的类型均可以由 φ 做替换得到。

类型推导算法

接下来我们将要给出一个算法 [2]，计算一个 Curry 风格 λ 项的主类型。算法首先给每个 λ 抽象的变量 x_i 赋予两两不同的类型变量 α_i，接下来在计算 λ 项类型的同时，维护一个“限制方程组”，其中每一项形如 $\varphi_i = \psi_i$，表示 φ_i 和 ψ_i “是相同的类型”。不妨讨论 λ 项的构成。

1. 若 λ 项为 x_i，此项的类型为 α_i
2. 若 λ 项为 $\lambda x_i.\, e$，递归计算 e 的类型 φ，此项的类型为 $\alpha_i \to \varphi$
3. 若 λ 项为 $(e_1\ e_2)$，递归计算 e_1 的类型为 φ_1，e_2 的类型为 φ_2，此项的类型为 ρ_i，且将限制 $\varphi_1 = \varphi_2 \to \rho_i$ 加入限制方程组。其中，ρ_i 是一个未使用过的新类型变量。

根据这一算法，我们求解出 λ 项的一个序对 (φ, C)，其中 φ 是其类型，C 是类型变量之间的一些“限制”。因而类型推导问题就转化成了求解“限制”的问题。更严谨地，一个限制方程组是一系列类型“等式” $\varphi = \psi$ 的集合，表明了在类型推断的过程中需要满足的限制。我们可以通过等量替换的方式化简限制方程组。

性质 1： 算法求得 λ 项的类型 φ 是其主类型的下界。

证明： 考虑主类型的类型指派，这是显然的。

性质 2： 限制方程组可以被转化为等价的限制方程组，使得

1. 限制方程组形如 $\{b_1 = \varphi_1, b_2 = \varphi_2, \ldots, b_k = \varphi_k\}$
2. 对于任意的 $i \le j$，$b_i \notin \mathrm{FV}(\varphi_j)$。换言之，我们将限制方程组转化成了一个“上三角”的形式。

证明： 首先需要明确“等价的限制方程组”的含义。对于一个限制方程组 C，由其定义的等价关系是满足以下规则的最小等价关系

1. 如果 $\varphi = \psi \in C$，那么 $\varphi = \psi$;
2. $\varphi_1 = \psi_1 \wedge \varphi_2 = \psi_2$ 当且仅当 $\varphi_1 \to \varphi_2 = \psi_1 \to \psi_2$。

我们称两个限制方程组等价，如果它们定义出的等价关系是相等的。转化算法类似解代数方程组的带入消元算法，具体而言，变换 $\mathrm{unify}(C \cup \{\varphi = \psi\})$ 定义为：

1. $\varphi = \beta$，且 $\beta \notin \mathrm{FV}(\psi)$，
   $\mathrm{unify}(C \cup \{\varphi = \psi\}) := \mathrm{unify}(C[\beta \mapsto \psi]) \cup \{\beta = \psi\}$，显然 β 一定不会出现在 $C[\beta \mapsto \psi]$;
2. $\psi = \beta$，且 $\beta \notin \mathrm{FV}(\varphi)$，和上一种情况类似
3. $\varphi = \varphi_1 \to \varphi_2$，且 $\psi = \psi_1 \to \psi_2$，那么
   $\mathrm{unify}(C \cup \{\varphi = \psi\}) := \mathrm{unify}(C \cup \{\varphi_1 = \psi_1\} \cup \{\varphi_2 = \psi_2\})$
4. 其他情况，失败，返回 $\varnothing$

很容易验证，变换的每一步均保持限制方程组等价。另一方面，由于每一次 1,2 均会使类型变量数量减小，3 仅能连续执行有限次，我们的算法总会在有限步内终止。

引理： 限制方程组可以被转化为等价的限制方程组，使得

1. 限制方程组形如 $\{b_1 = \varphi_1, b_2 = \varphi_2, \ldots, b_k = \varphi_k\}$

2. 对于任意的 i, j，$b_i \notin \mathrm{FV}(\varphi_j)$。换言之，我们将限制方程组转化成了一个“解”的形式。

证明： 首先根据性质 2 对限制方程组做上三角化，接下来从下至上，用替换操作消去左侧变量所有的出现。

算法： 我们的类型推导算法分为以下三步

1. 为所有变量分配两两不同的限制方程组，计算 λ 项的类型和限制方程组
2. 解限制方程组
3. 将所有限制 $b_i = \varphi_i$ 左侧的类型变量用右侧的类型替换

例 1： $\lambda x.\,\lambda y.\,x\ (x\ y)$，首先指派 $x : \alpha, y : \beta$，那么有

1. $\lambda x.\,\lambda y.\,x\ (x\ y) : \alpha \to \beta \to \sigma$，限制方程组为 $\{\alpha = \beta \to \rho, \alpha = \rho \to \sigma\}$
2. 解限制方程组，得到 $\alpha = \sigma \to \sigma, \beta = \sigma, \rho = \sigma$
3. 将类型按照解替换为 $(\sigma \to \sigma) \to \sigma \to \sigma$

例 2： $\lambda x.\,x\ x$，首先指派 $x : \alpha$，那么有

1. $\lambda x.\,x\ x : \alpha \to \rho$，限制方程组为 $\{\alpha = \alpha \to \rho\}$
2. 注意到限制方程组中存在一个环，解限制方程组失败

解限制方程组的复杂性

注意到上面的算法运行时间可能为指数级别。举例而言，限制方程组

$$
\begin{aligned}
a_1 &= a_2 \to a_2 \\
a_2 &= a_3 \to a_3 \\
&\cdots \\
a_n &= a_{n+1} \to a_{n+1}
\end{aligned}
$$

按照 $a_n, a_{n-1}, \ldots, a_1$ 的顺序消去变量，则最终类型长度高达 $O(2^n)$。但幸运的是，由于类型长度的增加是由替换引起的，对于类型对应的树构成的森林，其本质不同的子树数量并不会增加。因此我们可以将类型对应的树构成的森林转化为 DAG。对于 DAG 上的一个结点，其 BFS 生成的树是其对应的类型树。

在实现中，在类型树构成森林的基础上加一个基于并查集的表 $T : \mathrm{Node} \to \mathrm{Node}$，支持

- 将 $T[u]$ 设为 v
- 找到 $T^\infty[u]$，记作 $T(u)$

我们的算法可以描述如下

1. 初始时，为每一个限制 $\varphi = \psi$ 的两侧建立对应的类型树，并预处理 `T[u] = u`
2. `Unify(u, v)` 表示将结点 `u, v` 对应的类型树相等的限制，加入到数据结构（T+U）之中，那么

   a. 设 `u=T(u), v=T(v)`
   b. 如果 `u=v`，说明这一条限制是冗余的

c. 如果 `u` 对应于类型变量，设置 `T[u]=v`，并将 `u` 加入集合 S；如果 `v` 对应于类型变量，设置 `T[v]=u`，并将 `v` 加入集合 S。集合 S 对应了限制方程组解左侧的“约束变元”。
d. 如果 `u=u1->u2`，且 `v=v1->v2`，递归 `Unify(u1, v1), Unify(u2, v2)`
e. 否则，返回失败

3. 依次 `Unify` 所有的限制
4. `BFS(u)` 表示找到 `u` 对应的类型树，依次 `BFS` 所有集合 S 中的结点并产生对应的类型树。
   a. 如果任何一个结点 `BFS` 过程中回到了自身，返回失败
      i. 这是由于在算法第第二步 3 中，我们没有检查 $u \notin \mathrm{FV}(v)$
5. S 和 `BFS` 产生的类型树对应的类型构成了一组解

由于每一次 `Unify` 会将一对原类型树构成森林的子树判定为相同，而本质不同的对数仅有 $O(n^2)$ 个，因此 `Unify` 仅会执行 $O(n^2)$ 次，从而总的时间复杂度也是多项式级别的。

关于这一算法，[3] 给出了更详细的描述。

组合子逻辑

无类型组合子逻辑

组合子逻辑是为了消除 λ 演算中变量的存在。具体来说，我们希望找到一组基 $\{B_1, B_2, \ldots, B_n\}$ 满足任意 λ 项都可以被 B_i 和它们之间的组合表示。所谓表示，是指“外延相同”，即给足参数后计算的结果相同。事实上，一组可能的基是：

- $\mathbf{S} = \lambda xyz.\,(x\ z)\ (y\ z)$
- $\mathbf{K} = \lambda xy.\,x$

下面就来证明任何 λ 项都存在一个外延相同的 λ 项，使其为 $\mathbf{S}\ \mathbf{K}$ 组合子之间的应用。首先有

- $\mathbf{I} = \mathbf{S}\ \mathbf{K}\ \mathbf{K}$，其中对于所有 F，均有 $\mathbf{I}\ F \to_\beta \mathbf{K}\ F\ (\mathbf{K}\ F) \to_\beta F$

定义一个组合子项为 $\mathcal{C} := V \mid \mathbf{K} \mid \mathbf{S} \mid (\mathcal{C}\ \mathcal{C})$，其中 V 是变量集合，用来在将 λ 项转化为组合子项时的过渡。定义组合子项的 w 规约 $\to_w$ 为

$$
\begin{gathered}
\mathbf{K}\ F\ G \to_w F \\
\mathbf{S}\ F\ G\ H \to_w F\ H\ (G\ H) \\
\frac{F \to_w F'}{F\ G \to_w F'\ G} \quad \frac{F \to_w F'}{G\ F \to_w G\ F'}
\end{gathered}
$$

同理可定义 $\twoheadrightarrow_w$。定义组合子项的自由变量 $\mathrm{FV}(c)$ 为其中出现的所有变量。对于 $F \in \mathcal{C}$，定义 $\lambda^* x.\,F$ 为

$$
\begin{aligned}
\lambda^* x.\, x &:= \mathbf{I} \\
\lambda^* x.\, F &:= \mathbf{K}\, F, x \notin \mathrm{FV}(F) \\
\lambda^* x.\, F\, G &:= \mathbf{S}\, (\lambda^* x.\, F)\, (\lambda^* x.\, G)
\end{aligned}
$$

很容易说明 $\lambda^* x.\, F$ 是良定义的。正如符号指出的一样，施归纳于定义很容易说明

$$(\lambda^* x.\, F)\, G \to_w F[x \mapsto G]$$

下面考虑 λ 项到组合子项的转化算法，定义 $(*)_{\mathcal{C}} : \Gamma \to \mathcal{C}$ 为

$$
\begin{aligned}
(x)_{\mathcal{C}} &:= x \\
(e_1\ e_2)_{\mathcal{C}} &:= (e_1)_{\mathcal{C}}\ (e_2)_{\mathcal{C}} \\
\lambda x.\, e &:= \lambda^* x.\, (e)_{\mathcal{C}}
\end{aligned}
$$

非形式化地，K 组合子充当了放弃无用参数的作用，而 S 组合子充当了参数替换的作用。施归纳于生成算法容易说明，由 λ 项生成的组合子项和原先 λ 项是外延相同的，因此无类型组合子逻辑的表示能力和无类型 λ 演算等价，且均是图灵完备的。

简单类型组合子逻辑

类似简单类型 λ 演算，我们可以给每个组合子项赋予类型，例如 $\mathbf{I}$ 的主类型为 $\alpha \to \alpha$，$\mathbf{K}$ 的主类型为 $\alpha \to \beta \to \alpha$，而 $\mathbf{S}$ 的主类型为 $(\alpha \to \beta \to \eta) \to (\alpha \to \beta) \to \alpha \to \eta$。由此构成的系统称为 Curry 风格的简单类型组合子逻辑。类似地，也可以定义出 Church 风格的简单类型组合子逻辑。

由于 λ 演算和组合子逻辑间的对应关系，简单类型组合子逻辑也具有类似地 Church-Rosser 定理，正则定理等等。

Curry-Howard 同构

Curry-Howard 同构说明了类型系统和逻辑系统之间的重要联系。事实上我们可以说明

1. 简单类型 λ 演算对应于直觉主义谓词逻辑的自然演绎系统
2. 简单类型组合子逻辑对应于希尔伯特公理系统

在 Curry-Howard 同构的视角下，有

- 类型对应于逻辑命题：例如 $\alpha \to \beta \to \alpha$ 既可以看作 λ 项的类型，也可以看作谓词逻辑中的一个命题
- 程序对应于证明：一个类型为 φ 的程序对应于对命题 φ 的证明
    a. λ 应用 $e_1\ e_2$ 对应于证明规则 $\frac{P \to Q\ \ P}{Q}$
    b. λ 项的语法树对应于自然演绎系统的一棵证明树（在之后解释）
    c. 组合子项对应于公理系统中的一个证明
- λ 项的正则化对应于自然演绎中证明树的正则化

- ...

自然演绎系统

自然演绎系统（Natural Deduction System）是一种用来研究证明形式化的证明系统。在自然演绎系统中，一个证明是一棵以结论为根的有根树（通常来说，我们将叶子画在上方，将根画在下方），其所有的叶子结点是假设，且由以下的生成规则生成。

1. $\to$ 引入规则：给定一棵以 Q 为根的证明树，且树中存在一个“开”的假设 $[P]$，可以建立 $\frac{Q}{P\to Q}$，并将假设 P 关闭。为表述方便，上面的树记作 $[P]\ldots Q$；
2. $\to$ 消去规则：给定一棵以 $P\to Q$ 为根的证明树和一棵以 P 为根的证明树，可以建立 $\frac{P\ P\to Q}{Q}$；
3. $\land$ 引入规则：给定以 P, Q 为根的证明树，可以建立 $\frac{P\ Q}{P\land Q}$；
4. $\land$ 消去规则：给定以 $P\land Q$ 为根的证明树，可以建立 $\frac{P\land Q}{P}$ 或者 $\frac{P\land Q}{Q}$；
5. $\lor$ 引入规则：给定以 P 为根的证明树，可以建立 $\frac{P}{P\lor Q}$ 或者 $\frac{P}{Q\lor P}$；
6. $\lor$ 消去规则：给定以 $P\lor Q$，$[P]\ldots R$ 和 $[Q]\ldots R$ 为根的证明树，可以建立 $\frac{P\lor Q\ [P]\ldots R\ [Q]\ldots R}{R}$，并将 $[P],[Q]$ 关闭；
7. $\bot$ 消去规则：给定以 $\bot$ 为根的证明树，可以建立 $\frac{\bot}{P}$。这条规则是说，假设为矛盾，则可以得到一切结论。通常的 $\neg P$ 被定义为 $P\to\bot$。
8. **!** 双重否定律：给定以 $\neg\neg P$ 为根的证明树，可以建立 $\frac{\neg\neg P}{P}$ 为根的证明树。

一般来说，1-7 构成的系统称为直觉主义自然演绎系统 IPC，1-8 构成的系统称为经典自然演绎系统，而仅有 1,2,7 的系统称为 IPC($\to$)。很容易发现，IPC($\to$) 和 $\lambda_\to$ 有着对应关系，IPC 则与引入了 $\land,\lor,\bot$ 对应的类型构造子和类型规则的 λ 演算对应。我们可以证明，类型存在一个 λ 项，当且仅当类型对应的命题存在一棵证明树。

自然演绎系统可以写成所谓**串行演算（Sequential Calculus）**的形式。

希尔伯特公理系统

希尔伯特公理系统是另一种表示逻辑的形式系统。直觉主义的希尔伯特公理系统由唯一的分离规则（Detachment Rule）和两条公理构成。其中，分离规则是说 $\frac{P\ P\to Q}{Q}$，而两条公理是：

1. $P\to Q\to P$
2. $(P\to Q\to R)\to(P\to Q)\to P\to R$

很明显，两条公理对应于组合子 $\mathbf{S},\mathbf{K}$ 的类型，分离规则对应于组合子应用的类型规则。由于组合子逻辑和 $\lambda_\to$ 的对应，我们也得到了希尔伯特公理系统与 IPC($\to$) 证明能力的等价性。

附录

记号说明

- x, y, z, f, g, h 表示变量；
- e, o 表示任意 λ 项；
- $\varphi, \psi, \rho, \sigma$ 等表示类型；
- 使用 α, β 表示基本类型；
- 使用 $?$ 表示上下文；
- $\to_\beta$ 表示一步 β 规约，w 规约类似；
- $\twoheadrightarrow_\beta$ 表示多步 β 规约，w 规约类似；
- $\mathrm{FV}\ e$ 表示自由变量集合；
- $e[x \mapsto e']$ 表示将 e 中的自由变量 x 替换为 e'；
- Γ 表示 λ 项的集合，$\mathcal{C}$ 表示组合子项的集合。

致谢

感谢 @wmd 同学在作者学习中的帮助。如果您发现了本文中的问题，欢迎用评论或邮件方式联系我。

参考文献

1. An Introduction to Logical Relation. arXiv : Programming Languages, 2019. Lau Skorstengaard.↩

2. Lectures on the Curry-Howard Isomorphism. Studies in Logic and the Foundations of Mathematics, 2006. Morten Heine Sorensen and Pawel Urzyczyn. ↩

3. Compilers: Principles, Techniques, and Tools. Alfred V Aho, et al. ↩