简单类型化的 λ-演算

读书笔记

许博

1 引入类型

第一章中介绍的无类型 λ-演算的定义十分简洁而且优雅，但是有些时候过于自由，比如它允许 (xx) 这样没有意义的 λ-项。为了更好地掌握函数的期望行为，本章将引入类型。

函数通常被认为作用于某一集合的对象上，比如自然数的集合或者一条线上点的集合。

类型的添加提供了在允许的输入值上的一些限制，比如定义在自然数域上的函数只能接受自然数作为输入值，即使对于非法的输入值的可能的输出值是清楚的。比如定义在自然数域上的立方函数，不能计算 3.5 的立方值，只能定义另一个作用于更大的域，比如实数域的立方函数。

本章中引入的简单类型形式化了第一个重要的步骤，尽管某些时候限制性过强：我们不能通过简单类型来表示足够多的函数。后续章节将会增加更多的类型来加强系统的表达能力。

2 简单类型

添加（抽象）类型的一个直接的方式是从一个类型变量的无限集合出发，然后添加乘法规则来构建更复杂的类型，即函数类型。

记类型变量的无限集合：$\mathbb{V} = \{\alpha, \beta, \gamma, ...\}$。

定义 2.1 所有简单类型的集合 $\mathbb{T}$

(1)（类型变量）如果 $\alpha \in \mathbb{V}$，则 $\alpha \in \mathbb{V}$。

(2)（箭头类型）如果 $\sigma, \tau \in \mathbb{T}$，则 $(\sigma \to \tau) \in \mathbb{T}$。

抽象语法描述为：$\mathbb{T} = \mathbb{V} | \mathbb{T} \to \mathbb{T}$。

记号 2.2 *(1)* 希腊字母 $\alpha, \beta, ...$ 以及它们的变体用于表示 $\mathbb{V}$ 中的类型变量。

(2) 使用 $\sigma, \tau, ...$（偶尔使用 $A, B, ...$）来表示任意的简单类型。

(3) 最外层的括号会被省略。

(4) 箭头类型中的括号是右结合的。

$\alpha_1 \to \alpha_2 \to \alpha_3 \to \alpha_4$ 是 $(\alpha_1 \to (\alpha_2 \to (\alpha_3 \to \alpha_4)))$ 的简写。

使用形如 $M : \sigma$ 的格式表示项 M 具有类型 σ。

假设对于每个类型 σ，都有无限的变量可以获得，而每一个变量都只有一个唯一的类型：如果 $x : \sigma \wedge x : \tau$，则 $\sigma \equiv \tau$。

类型化应用和抽象所需要的必要条件如下：

(1)（应用）：如果 $M : \sigma \to \tau \wedge N : \sigma$，则 $MN : \tau$。

(2)（抽象）：如果 $x : \sigma \wedge M : \tau$，则 $\lambda x.M : \sigma \to \tau$。

定义 2.3（可类型化的项，*typable term*）如果存在类型 σ 使得 $M : \sigma$，则项 M 称为可类型化的（*typable*）。

3 邱奇-类型化（Church-typing）和柯里-类型化（Curry-typing）

类型化一个 λ-项从类型化它的变量开始，有两种方式给出变量的类型：

(1) 在每一个变量的声明中，显式的指定类型。这种方式叫做显式类型化。如果考虑到在确定应用的类型时的限制，更复杂的项的类型也可以直接确定（我猜这里的意思是假设所有的应用中左右项的类型都是满足限制的）。

(2) 另一种方式不给出变量的类型，在一定范围内隐式的确定变量的类型。这种方式叫做隐式类型化，通过一个查找过程来推断类型，其中可能包含了猜测。

为了清楚的表示，约束变量的类型会在 λ 后引入它们（对应的绑定变量）时直接标记，自由变量的类型则会在所谓的上下文（context）中给出，顺序随意。

如 $x : \alpha, y : (\alpha \to \alpha) \to \beta \vdash (\lambda z : \beta.\lambda u : \gamma.z)(yx) : \gamma \to \beta$，读作在上下文 $x : \alpha, y : (\alpha \to \alpha) \to \beta$ 中，项 $(\lambda z : \beta.\lambda u : \gamma.z)(yx) : \gamma \to \beta$ 具有类型 $\gamma \to \beta$。分隔符 $\vdash$ 分割上下文和可类型化的项。

4 $\lambda\rightarrow$ 的推导规则

现在 λ-项具有类型信息，给出预先类型化的 λ-项的新的定义，新的集合也被记为 $\Lambda_{\mathbb{T}}$：

定义 4.1（预先类型化的 λ-项，$\Lambda_{\mathbb{T}}$）

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

为了表示形如 λ-项 M 在上下文 Γ 中具有类型 σ'，进行定义：

定义 4.2（语句，声明，上下文，判决）

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

(2) 一个声明是一个变量作为主体的语句。

(3) 一个上下文是具有不同主体的声明的列表。

(4) 一个判决形如 $\Gamma \vdash M : \sigma$，其中 Γ 是一个上下文，$M : \sigma$ 是一个语句。

给出能够判断一个判决 $\Gamma \vdash M$ 是否是可推导的，也即 M 在上下文 Γ 中具有类型 σ 的形式化规则。

给出的规则形式是一个所谓的推导系统（derivation system）：每一个规则解释某种判决如何能够被形式化确定。三个推导规则中的每一个都是所谓的前提-结论格式，一些前提出现在水平线之上，结论在水平线下边：

$$\frac{\textbf{premiss 1} \quad \textbf{premiss 2} \quad \ldots \quad \textbf{premiss n}}{\textbf{conclusion}}$$

这个推导结构的含义是：在已知 **premiss 1**，**premiss 2**，... ,**premiss n** 的情况下。可以推出 **conclusion**。当前提的个数为 0 时，则只需要写结论即可，同时无需水平线。

给出 $\lambda\rightarrow$ 的三条推导规则，这三条规则为 $\lambda\rightarrow$ 形式化了一个推导系统：

定义 4.3 $\lambda\rightarrow$ 的推导规则

（变量）如果 $x : \sigma \in \Gamma$，则 $\Gamma \vdash x : \sigma$

（应用）$\dfrac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}$

（抽象）$\dfrac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x : \sigma.M : \sigma \to \tau}$

定义 4.4 合法的 $\lambda\!\rightarrow$-项

如果存在上下文 Γ 以及类型 ρ 使得 $\Gamma \vdash M : \rho$，则在 $\lambda\!\rightarrow$ 中的预先类型化的项 M 称为合法的。

5 $\lambda\!\rightarrow$ 中推导的不同格式

树型，线型以及旗型（一种特殊的线型，更加精简）。

6 类型理论中需要解决的问题的种类

三种：

(1) Well-typedness，也叫 Typability。我的理解是良好的类型定义，类型定义的信息是足够的：

$? \vdash term : ?.$

如果一个项是合法的，能够找到一个合适的上下文和项的类型，如果不是合法的，也能展示出为什么是错误的。

(1a) (1) 的一个变体也即 $TypeAssignment$，类型赋值，在给出上下文的时候，找到项的类型：

$context \vdash term : ?.$

(2) 类型检查：

$context \vdash^{?} term : type$，在给定上下文，项以及它的类型时，检查在给定的上下文时，给定的项是否具有给定的类型。

(3) 项的查找：

$context \vdash ? : type.$

给定上下文和项的类型，找到是否存在一个项在给定的上下文中具有给定的类型。

7 项的查找

PAT-解释，PAT 意为 propositions-as-types 以及 proofs-as-terms，查找在上下文中符合给定类型的 $\lambda\!\rightarrow$-项，整个过程可以解释为等价的逻辑证明。

8 $\lambda\!\to$ 的一些性质

首先是一些定义：

定义 8.1（域，dom，子上下文，$\subseteq$，排列，投影，$\upharpoonright$）

(1) 如果 $\Gamma \equiv x_1 : \sigma_1, ..., x_n : \sigma_n$，则 Γ 的域或 $dom(\Gamma)$ 是列表 $(x_1, ..., x_n)$。

(2) 如果所有出现在 Γ' 的声明以同样的顺序出现在 Γ 中，则上下文 Γ' 是 Γ 的一个子上下文，或 $\Gamma' \subseteq \Gamma$。

(3) 如果所有出现在 Γ' 的声明出现在 Γ 中，如果所有出现在 Γ 的声明也出现在 Γ' 中，则 Γ' 是 Γ 的一个排列（*permutation*），反之亦然。

(4) 如果 Γ 是一个上下文，且 Φ 是变量的一个集合，则 Γ 在 Φ 上的投影，或记为 $\Gamma \upharpoonright \Phi$，是 Γ 的子上下文，记为 Γ'，满足条件 $dom(\Gamma') = dom(\Gamma) \cap \Phi$。

性质：

引理 8.2（自由变量引理）

如果 $\Gamma \vdash L : \sigma$，则 $FV(L) \subseteq dom(\Gamma)$。

引理 8.3（*Thinning*，*Condensing*，*Permutation*）

*(1) (*稀疏引理，*Thinning)* 令上下文 Γ' 和 Γ'' 满足条件 $\Gamma' \subseteq \Gamma''$，如果 $\Gamma' \vdash M : \sigma$，也有 $\Gamma'' \vdash M : \sigma$。

*(2) (*压缩引理，*Condensing)* 如果 $\Gamma \vdash M$，也有 $\Gamma \upharpoonright FV(M) \vdash M : \sigma$。

*(3) (*排列引理，*Permutation)* 如果 $\Gamma \vdash M : \sigma$ 且 Γ' 是 Γ 的一个排列，则 Γ' 也是一个上下文且 $\Gamma' \vdash M : \sigma$。

引理 8.4（生成引理，*Generation Lemma*）

(1) 如果 $\Gamma \vdash x : \sigma$，则 $x : \sigma \in \Gamma$。

(2) 如果 $\Gamma \vdash MN : \tau$，则存在类型 σ 使得 $\Gamma \vdash M : \sigma \to \tau$ 且 $\Gamma \vdash N : \sigma$。

(3) 如果 $\Gamma \vdash \lambda x : \sigma.M : \rho$，则存在 τ 使得 $\Gamma, x : \sigma \vdash M : \tau$ 且 $\rho \equiv \sigma \to \tau$。

生成引理从构造的角度解释 judgement 是如何产生的。可以看作是推导规则的逆过程。

引理 8.5（子项引理）如果 M 是合法的，则 M 的每一个子项都是合法的。

引理 8.6（唯一性引理）假设 $\Gamma \vdash M : \sigma$ 且 $\Gamma \vdash M : \tau$，则 $\sigma \equiv \tau$。

唯一性引理也即在给定的上下文中，任意给定的项都只有唯一的类型。

9 规约和 $\lambda\rightarrow$

引理 9.1（替换引理）假设 $\Gamma', x:\sigma, \Gamma'' \vdash M:\tau$ 且 $\Gamma' \vdash N:\sigma$，则 $\Gamma', \Gamma'' \vdash M[x:=N]:\tau$。

再替换时，x 被替换为同类型的项，而 M 的类型不发生改变。