λ 演算中不动点组合子的个数是无穷的

孙启翔

November 2024

摘要

λ 演算是函数式编程语言的模型。不动点组合子 (fixed point combinator，缩写为 fpc) 是 λ 演算中非常重要的一部分，它可以实现对一个匿名函数的递归，以及找出任一高阶函数的不动点。而找到无穷个不动点组合子可以帮助我们处理递归时采取多样的方式，编写更加可靠的代码。

本文主要论证我们可以找到无穷个不动点组合子，并简单阐释找到它的启发，即从 Curry's fpc、Turing's fpc 出发，联系到 β-归约，试图阐明发现无穷多个不动点组合子的思路。

1 引言

在 λ 演算中，函数被视作是 "一等公民"，可以被当作参数传入给其他函数，也可以作为结果从函数体里返回。fpc 是 λ 演算中一种特殊的高阶函数，它能够计算出任意一个函数的不动点。

通过 fpc，我们可以在不知道函数名的情况下调用该函数，这对实现递归十分重要。具体来说，在 λ 演算中，函数的定义是匿名的，我们无法直接通过函数名称调用函数，所以我们需要 fpc 来帮助我们调用函数。

那我们为什么要关注 fpc 的个数?

如果存在无穷多个 fpc，就可以使递归和自引用的实现方式在理论上具有多样性。同时不同的 fpc 可能在效率、适用性和表达方式上有所不同，这为研究者提供了探索不同递归实现方法的空间。

本文指出了一种构造无穷多个 fpc 的方法，并给出了它的证明过程以及发现它的思路。

2 无穷多个 fpc 的构造方式

我们要构造的 fpc 记为 Γ, 令$\Gamma = \gamma\gamma...\gamma (n \geq 2)$, 其中 Γ 为 γ 重复 n 次，$\gamma = \lambda a_1 a_2 \ldots a_{n-1} f.f(wf)$, 这里 w 是一个长度为 n、由 $a_1, a_2, \ldots, a_{n-1}$ 构成的一个表达式。例如:

$$\Gamma_{fpc} = (L\ L\ L\ \cdots\ L) \quad \text{(L 重复 26 次)}$$

$$L = \lambda abcdefghijklmnopqrstuvwxyzr.r(\text{thisisafixedpointcombinator})$$

下面我们将在第三节中证明 Γ 是一个 fpc

3 证明

对于第二节构造的 fpc Γ, 我们将以 $n = 2$ 为例论证 Γ 是一个 fpc。

当 $n = 2$ 时，

根据定义, 将构造出的 fpc 记为 Γ，则有

$$\Gamma = (\lambda xf.f(xxf))(\lambda xf.f(xxf))$$

我们要证明对任意函数 f，都有 $\Gamma F = F(\Gamma F)$, 我们计算 ΓF:

$$\Gamma F = (\lambda xf.f(xxf))(\lambda xf.f(xxf))F$$

根据 β-归约，我们将第二个 $(\lambda xf.f(xxf))$ 替换成第一个 $(\lambda xf.f(xxf))$ 中的 x，将 F 替换成第一个 $(\lambda xf.f(xxf))$ 中的 f，我们可以得到:

$$\Gamma F = F((\lambda xf.f(xxf))(\lambda xf.f(xxf))F)$$

我们可以发现，由于 ΓF还等于$(\lambda xf.f(xxf))(\lambda xf.f(xxf))F$，所以我们便得到了：

$$\Gamma F = F(\Gamma F)$$

我们也就证明出了 Γ 在 $n = 2$ 时是 fpc

依次类推我们可以得到对于 $n > 2$ ，Γ 也符合一个 fpc 的定义。

4 思路来源

4.1 Curry's fpc

Curry 发现的 fpc 也被称为 Y 组合子 (Y combinator)，它由 Haskell B. Curry 发现。Y 组合子的表达式可以写作

$$Y = \lambda f.(\lambda x.f(xx))(\lambda x.(f(xx))).$$

它是根据以下步骤构造出来的：

我们想要找出一个 fpc Y，它能为任意以 λ 形式定义的函数 F 提供一个不动点 X，满足 $F(X) = X$。

尝试构造这样的 Y 满足 $YF = F(YF)$，这样 YF 就是我们想要的不动点，并且它将适用于每一个 F。

也就是说，我们期望找出这样一个 ΩF，使得 ΩF 可以进行 β-归约化成 $F(\Omega F)$。

假定 $\Omega F = \omega F \omega F$，其中第一个 ωF 代表着 “控制”，第二个 ωF 代表着 “复制”。

具体一点说，第一个 ωF 负责保证函数的递归调用能正确进行，第二个 ωF 负责复制原始的 ωF。当第一个 ωF（控制）将 F 应用到第二个 ωF 时，第二个 ωF 会返回自身，这样就可以不断地将 F 应用到自身，实现递归调用的效果。

因此，我们需要 ωFx 可以 β-归约到 $F(xx)$ 的形式。令 $\Omega F = \lambda x.F(xx)$，这样我们便得到了 Curry's fpc:

$$Y = \lambda f.\Omega f \Omega f = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

4.2 Turing's fpc

对于 Curry's fpc，任取 F，我们并不能通过 β-归约将 YF 化简成 $F(YF)$ 这种形式。而我们将要介绍的 Turing's fpc 则可以较为明确地进行化简，它的构造方式和 Curry 的 fpc 很类似:

我们不妨设 Turing's fpc 表示为 Y_1，那么 Y_1 的表达可以写作

$$Y_1 = \eta\eta = (\lambda xf.f(xxf))(\lambda xf.f(xxf))$$

不难发现，这正是我们在第 2 节中构造的 fpc 的 $n = 2$ 时的特殊形式

下面是它具体的构造方法:

令 $Y_1 = \eta\eta$，使得 $Y_1F = \eta\eta F$ 可以 β-归约成 $F(\eta\eta F)$ 的形式。

类似于 Curry's fpc 的构造，我们令第一个 η 负责 "控制"，第二个 η 负责 "复制"，所以我们需要 ηxf 可以 β-归约成 $f(xxf)$ 的形式。

我们只需要令 $\eta = \lambda xf.f(xxf)$，此时我们便得到了 Turing's fpc

$$Y_1 = (\lambda xf.f(xxf))(\lambda xf.f(xxf))$$

4.3 Turing's 与 Curry's fpc 为什么是两个不同的 fpc

我们不妨假定存在两个 fpc 分别记为 Y_1 和 Y_2，如果 Y_1 经过有限次 β-归约可以变换成 Y_2，那么我们认为这两个 fpc 是等价的。

下面我们将要证明 Turing's fpc 和 Curry's fpc 并非等价:

我们令 Y_1 和 Y_2 分别为 Curry's fpc 和 Turing's fpc

$$Y_1 = \lambda f.\Omega f \Omega f = \lambda f.(\lambda x.f(xx))(\lambda x.(f(xx))).$$

$$Y_2 = \eta\eta = (\lambda xf.f(xxf))(\lambda xf.f(xxf))$$

只需证 Y_1 和 Y_2 可以进行 β-归约，即 $\lambda f.\Omega f\Omega f$ 与 $\eta\eta$ 可以进行 β-归约。

所以只需证 $\lambda f.\Omega f\Omega f$ 与 $\lambda f.f(\eta\eta f)$ 可以进行 β-归约。

等价于证 $\Omega f\Omega f$ 可以与 $f(\eta\eta f)$ 可以进行 β-归约。

等价于 $\Omega f\Omega f$ 可以与 $\eta\eta f$ 进行 β-归约。

等价于 $f(\Omega f\Omega f)$ 可以与 $f(\eta\eta f)$ 进行 β-归约，而这绕回了我们原本要证明的东西，所以我们可以知道我们无法证 Turing's fpc 与 Curry's fpc 可以经过 β-归约化归到同一形式。那我们就可以认为这是两个不同的 fpc。

5 结论

本文在第二节构造了一种创造无穷个 fpc 的方法，并在第三节中对其中一种特殊情况做了证明，说明这种构造方式确实构造出来的都是 fpc。

接着在第四节中提到了 Curry's fpc 和 Turing's fpc 的构造方式以及验证这两个 fpc 本质上不是等价的。

但本文在第二节提出的构造方法仍然不能说明 fpc 的个数是无穷的，因为通过这种方式构造出来的 fpc 有可能是等价的，这还需要后续的工作进行研究。