Y Combinator 是怎么来的?

当你看到这个奇怪的表达式时, 你肯定会问: 这哪里来的??

```
Y = λ p . (λ f . p (f f)) (λ f . p (f f))
```

本补充材料提供一个视角.

假设我们要写一个递归函数 `sum`, 满足 `sum n = 0 + 1 + 2 + ... + n`. 同时我们假设已经有了 `is_zero` 函数判断一个自然数是否是 0, 以及一个 `pred` 函数求出一个自然数的前驱 (5 的前驱是 4, 0 的前驱没有定义).

我们希望能写

```
sum = λ n . is_zero n 0 (n + sum (pred n))
```

然而, 本质上说所有的 λ-term 就是 λ 绑定, 函数作用和变量组成的, 它并不提供这种自我引用名字的机制 (考虑一下, 上面的这个等式能写成一个纯粹的 λ-term 吗? 目前写不成).

那该怎么办? 我们现在遇到的问题是, 我们只能定义函数, 函数体里只能用函数的参数, 不能用函数的名字 (或者说函数根本就没有名字). 一个巧妙的解决方法是这样的, 我们用 Python 来说明.

```
def sum(n):
    if n == 0:
        return 0
    else:
        return n + sum(n - 1)
```

我们想要这样的 `sum` 函数, 但 λ-calculus 里没有机制直接做这样的定义. 下面来想办法.

```
def proto(f, n):
    if n == 0:
        return 0
    else:
        return n + f(f, n - 1)
```

我们先如上定义一个 `proto` 函数, 它比 `sum` 多一个参数 `f`, 放在最前面. 我们本来要调用 `sum(n - 1)` 的位置现在写 `f(f, n - 1)`. 为什么写这个? 马上揭晓, 但现在主要是明确: `proto` 这个函数里没有引用 `proto` 这个名字本身, 里面只用了 `f` 和 `n` 这两个参数. 这是合法的, 可以被表达为 λ-term 的东西.

好, 现在想这件事: 我调用 `proto(proto, 5)` 会怎么样? 是不是就会执行到 `else` 那里, 然后 `return 5 + f(f, 4)` 就变成了 `return 5 + proto(proto, 4)`, 因为我们传进去的第一个参数是 `proto`, 所以 `f` 就带入了 `proto`. 诶! `proto(proto, 4)` 一样如此, 会 `return 4 + proto(proto, 3)`, 一直这样下去, 直到 `return 1 + proto(proto, 0)`, 而 `proto(proto, 0)` 直接返回了 0. 这样我们的递归就完成了!

抽象出来, 就是发生了如下的很神奇的事情. 我们定义

```
u = λ f . ... f x y z ...
```

然后定义

```
proto = λ f . u (f f)
      = λ f . (λ f . ... f x y z ...) (f f)
      = λ f . ... f f x y z ...
```

就会出现

```
proto proto = (λ f . ... f f x y z ...) proto
            = ... proto proto x y z ...
```

这样 `proto proto` 就成了我们的递归函数了!

回到 `sum` 的例子上,

```
u = λ f . λ n . is_zero n 0 (n + f (pred n))

proto = λ f . λ n . is_zero n 0 (n + f f (pred n))

proto proto = (λ f . λ n . is_zero n 0 (n + f f (pred n))) proto
            = λ n . is_zero n 0 (n + proto proto (pred n))

sum = proto proto
    = λ n . is_zero n 0 (n + sum (pred n))
```

正是我们想要的!

我们把整个过程打包起来. 最开始我们有 `u`, 最后我们要的是 `proto proto`, 那么

```
sum = proto proto
    = (λ f . u (f f)) (λ f . u (f f))
    = ( λ v . (λ f . v (f f)) (λ f . v (f f)) ) u
    = Y u
```

我们只是把这个打包的过程记作 `Y`:

```
Y = λ v . (λ f . v (f f)) (λ f . v (f f))
```