EOPL Pset 4

Alireza Habibzadeh `99109393`

4

الف

ابتدا ثابت می‌کنیم تابع `times` همان ضرب دو عدد است. در تعریف این تابع ابتدا تابع `maketimes` صدا زده شده. به عنوان ورودی `maker` به آن خود `maketimes` داده شده است. در بدنه‌ی تابع `maketimes` نیز دقیقا همین اتفاق افتاده پس از این به بعد می‌توانیم فرض کنیم همیشه `maker = maketimes` . اینجا انگار خود `maker` برای این به تابع داده شده تا بتواند از خودش در خودش استفاده کند بدون این که مستقیما چیزی از توابع بازگشتی بداند. ولی اتفاقی که در عمل می‌افتد این است که تابع یک تابع کپی خودش را در خودش استفاده می‌کند که دقیقا همان کار تابع بازگشتی است. در واقع با این کار در زبانی که از توابع بازگشتی به طور پیشفرض پشتیبانی نمی‌کند توابع بازگشتی تعریف کردیم که بسیار جالب است.

تابع `maketimes` با ورودی اولی که خودش باشد به زبان ریاضی این کار را با دو ورودی بعدی خود می‌کند:

$$maketimes(maketimes)(x)(y) = \begin{cases} 0 & \text{if } x=0 \\ maketimes(maketimes)(x-1)(y)-(0-y) & \text{if } x\neq 0 \end{cases}$$

که این تعریف بازگشتی ضرب دو عدد در یک دیگر است. پس این تابع `times` همان طور که از اسمش هم پیدا است، دو عدد صحیح را در هم ضرب می‌کند. (برای اعداد غیر صحیح در لوپ بی‌نهایت گیر می‌کند چون هیچ گاه `x` به صفر نمی‌رسد و منفی می‌شود.)

حال با شهودی که به این گونه ساخت توابع بازگشتی داریم، تابع `f` هم دقیقا شبیه همین کار را می‌کند منتهی هر بار مقدار خود تابع برای `x - 1` در مقدار فعلی `times` می‌شود. نتیجه تعریف بازگشتی تابع فاکتوریل است.

$$f(f)(5) = times(5)(f(f)(5-1)) = \ldots = 5! = 120$$

ب

```
let makeprime? = proc (maker)
                  proc (counter)
                   proc (n)
                    if zero? -(n, counter) ; counter =? n
                    then #t
                    else (and
                          (not (zero? (modulo n counter)))
                          (((maker maker) -(counter, -(0, 1))) n))
in let prime?  = ((makeprime? makeprime?) 2) ; start the counter from 2 (to n-1)
in (prime? 41)
```