Nome: Gilson Trombetta Magro

Matrícula: 18202192

Disciplina: INE5416 - Paradigmas de Programação

Turma: 04208 (2020.1)

1. QUESTÃO 8

Pesquise sobre o Combinador Y. O que é? O que ele faz? Descreva um pouco seu funcionamento.

2. RESPOSTA

Em cálculo-λ, um combinador é uma função que não possui variáveis independentes em seu escopo. O combinador Y é uma função de alta-ordem que recebe uma função f não-recursiva e torna f recursiva. Ele pode ser usado para implementar recursão em linguagens que não dão suporte à recursão explícita.

O combinador Y também é chamado de combinador de ponto fixo, porque usa a ideia de "ponto fixo" de funções em sua definição. Um ponto fixo para uma função f(x) é um valor x tal que x = f(x), isto é, um valor que permanece constante sob iteração. Um combinador Y (ou de ponto fixo), portanto, é uma função que satisfaça a seguinte equação: **Y f = f (Y f)** para todo f.

A implementação do combinador Y em cálculo-λ (considerando *lazy-evaluation*) é a seguinte:

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

ou, alternativamente, porém de forma equivalente:

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

A ideia aqui é utilizar uma função parcial para construir uma função geral recursiva. Tomemos o exemplo da função fatorial; podemos definir uma função "quase-fatorial" capaz de gerar uma função que calcula o fatorial de n, dada uma outra função "fatorial" que calcule o fatorial de n - 1:

```
quase-fatorial = lambda fatorial: lambda n: 1 if n == 0 else n * fatorial(n - 1)
```

contudo, não possuímos essa função “fatorial” que calcula o fatorial de n - 1, então dizemos que a função “quase-fatorial” geral uma outra função que calcula corretamente apenas o fatorial para n igual à zero:

```
quase-fatorial(<função qualquer>)(0) ⇒ 1
quase-fatorial(<função qualquer>)(n) ⇒ ?, com n > 0
```

Se agora aplicarmos a função quase-fatorial passando como parâmetro ela mesma, teremos:

```
quase-fatorial(quase-fatorial)(0) ⇒ 1
quase-fatorial(quase-fatorial)(1) ⇒ 1
quase-fatorial(quase-fatorial)(n) ⇒ ?, com n > 1
```

```
quase-fatorial(quase-fatorial(quase-fatorial))(0) ⇒ 1
quase-fatorial(quase-fatorial(quase-fatorial))(1) ⇒ 1
quase-fatorial(quase-fatorial(quase-fatorial))(2) ⇒ 2
quase-fatorial(quase-fatorial(quase-fatorial))(n) ⇒ ?, com n > 2
```

Portanto, observa-se que cada vez que a função quase-fatorial é aplicada sobre si mesma, ela gera uma função capaz de calcular corretamente o fatorial de um número a mais. Se repetirmos esse processo *ad infinitum*, no limite teremos a função fatorial completa. É aí que entra o combinador Y, que permite que façamos isso:

```
quase-factorial = lambda fatorial: lambda n: 1 if n == 0 else n * fatorial(n - 1)
Y = lambda f: (lambda x: x(x))(lambda y: f(lambda *args: y(y)(*args)))
fatorial = lambda n: Y(quase-fatorial)(n)
```

É importante, contudo, que sejam feitas considerações sobre *lazy-* e *strict-evaluation*, pois a definição do combinador Y muda dependendo de como a linguagem interpreta e resolve as expressões.

3. REFERÊNCIAS

- https://mvanier.livejournal.com/2897.html
- https://rosettacode.org/wiki/Y_combinator
- https://pt.wikipedia.org/wiki/Combinador_de_ponto_fixo