Resumo Avaliação por Demanda

Lourenço Bogo

29 de dezembro de 2020

Sumário

1 Introdução

O segundo tópico abordado na matéria foi "Lazy Evaluation"(Avaliação por Demanda).

Avaliação por demanda é uma estratégia de avaliação cujo princípio básico é: apenas calcular o valor de uma expressão quando esse valor for necessário. É o contrário do que vínhamos aprendendo até agora, que era avaliação ansiosa, cujo princípio é avaliar uma expressão na primeira vez que ela for encontrada.

Para ilustrar a diferença entre esses dois métodos de avaliação, segue um exemplo simples:

```
(cons (+ 1 2) '())
```

O código acima monta uma lista cujo primeiro elemento é a aplicação da função + nos elementos 1 e 2. Se estamos em uma linguagem onde temos avaliação ansiosa, acontecerá o seguinte: ao encontrarmos a operação `(+ 1 2)`, iremos avaliá-la e teremos como resultado 3. Nosso código então irá produzir uma lista com apenas 1 elemento, que será o número 3.

Fica claro que não usamos o valor 3 para nada, ele não foi necessário para nenhuma operação. Para montarmos a lista, não precisávamos saber que ao avaliarmos aquela expressão teríamos como resultado 3. Para isso que serve a avaliação por demanda. No código acima, se estivermos em uma linguagem que implementa esse tipo de avaliação, não iremos calcular o valor da soma, iremos criar uma lista de 1 elemento, cujo valor é uma **SUSPENSÃO**, que quando avaliada irá nos retornar o valor 3.

2 Suspensões

Suspensões são uma estrutura semelhante a um fechamento sem argumentos, elas guardam uma expressão e um ambiente. Quando o valor da suspensão for necessário, iremos interpretar a expressão dessa suspensão, com o ambiente contido nela.

Por eficiência, sempre que avaliamos pela primeira vez uma suspensão específica, substituímos no ambiente global o seu valor pelo valor retornado dessa avaliação. Assim, na próxima vez que precisarmos do valor dessa suspensão, poderemos utilizá-la sem ter que recalcular o seu valor.

Para todo esse sistema funcionar, necessitamos de um novo tipo de funções que serão chamadas **Funções Estritas**. Essas funções irão avaliar seus argumentos imediatamente, ou seja, caso recebam uma suspensão como argumento, elas irão expandir essa suspensão. Exemplificando:

```
(if (equal? (+ 1 2) 3) (alguma_coisa) (outra_coisa))
```

Nesse caso, para podermos continuar o programa, é necessário que avaliemos o valor da condição do if imediatamente. Ou seja, a condição do if é estrita, significando que ela nunca irá criar suspensões novas, e sempre irá avaliar as suspensões dadas.

Alguns outros exemplos de funções estritas são:

- Operações aritméticas, já que precisamos saber em quais valores estamos aplicando essa operação (não faz sentido somar duas suspensões)
- Car e cdr, já que quando queremos um elemento de uma lista, queremos o elemento em si e não uma suspensão

3 Vantages e Desvantagens

Primeiro, a grande desvantagem da avaliação por demanda é a seguinte: já que uma suspensão guarda também um ambiente, dependendo do jeito que implementarmos esse sistema, pode ser que o custo de memória fique muito alto e imprevisível. Além disso, funções com efeitos colaterais podem quebrar as coisas, já que podemos alterar suspensões antes de termos usado seu valor para o que queríamos.

Agora, as vantagens principais desse método de avaliação são:

- Aumento na performance da linguagem, já que iremos evitar avaliações desnecessárias
- Podemos ter estruturas de dados infinitas, pois só iremos calcular os elementos necessários dessa estrutura. Exemplo em haskell:

```haskell
fibs = 0 : 1 : zipWith (+) fibs (tail fibs)
      -- Código retirado de wiki.haskell.org/The_Fibonacci_sequence
      -- Nesse exemplo montamos uma lista que contém TODOS os números da
      -- sequência de Fibonacci. O único problema com isso é que se pedirmos algo que
      -- exige a lista toda (como o tamanho da lista), o programa irá quebrar.
```