Avaliação Preguiçosa

Carlos Lopes

Avaliação Preguiçosa

- A forma como as expressões são calculadas em HASKELL é denominada de avaliação preguiçosa.
- Neste mecanismo de avaliação uma expressão é somente calculada se é certo que o seu valor é necessário para obter o resultado.
- O oposto da avaliação preguiçosa é denominado de avaliação gulosa.
- Numa avaliação gulosa todos os argumentos da função devem ser calculados antes de se calcular o resultado da função.

Avaliação Preguiçosa

- Para melhor entender o processo de avaliação gulosa considere a função fun definida a seguir:

```
fun :: Int -> Int -> Int

fun a b = a+b
```

- Para avaliar a função fun aplicada a argumentos a_1 e a_2 é preciso substituir as variáveis correspondentes na definição da função pelas expressões a_i. Isto implica que a avaliação de

```
fun (1+2) (3-4)
```

é igual a:

```
(1+2)+(3-4)
```

Avaliação Preguiçosa

- As expressões `(1+2)` e `(3-4)` não são calculadas antes que sejam passadas para a função fun. Para o exemplo anterior, se desejarmos que a avaliação continue, deve-se avaliar os argumentos de +. Isto irá gerar a seguinte seqüência de cálculo:

```
fun (1+2) (3-4)
```

➔ `(1+2)+(3-4)`

➔ `3 + (3-4)`

➔ `3 + (-1)`

➔ `2`

Avaliação Preguiçosa

- Neste exemplo, as duas expressões `(1+2)` e `(3+4)`, que substituem as variáveis `a` e `b`, são calculadas. Entretanto, podem existir situações em que isto não acontece. Veja a definição da função fun2 descrita a seguir.

```
fun2 :: Int -> Int -> Int

fun2 a b = a+1
```

- Portanto, a expressão `(3+4)` correspondente a variável b não precisará ser calculada, pois ela não aparece na expressão à direita do sinal de =. Esta é a primeira vantagem da avaliação preguiçosa.

Avaliação Preguiçosa

- Outro exemplo:

```
fun3 :: Int -> Int  -> Int -> Int
fun3 a b c
          | a >= 0 = b
          | otherwise = c
```

- Vejamos agora o seguinte exemplo:

```
fun4 :: Int -> Int -> Int
fun4 a b = a*a
```

Se avaliarmos `fun4 (1+2) (4+3*2)` esta expressão será reduzida para:

```
(1+2) * (1+2)
```

Avaliação Preguiçosa

- Esta expressão poderia conduzir-nos a acreditar que a expressão `(1+2)` seria calculada duas vezes.
- Entretanto, isto não acontece. O uso de avaliação faz com que um argumento duplicado seja avaliado uma única vez. A seqüência da avaliação seria a seguinte:

```
fun4  (1+2) (4+3*2)
```

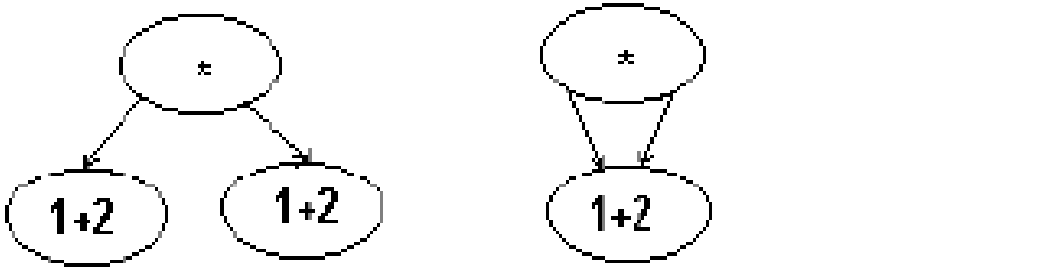
➔ `(1+2) * (1+2)`

➔ `3*3`

➔ `9`

Avaliação Preguiçosa

- Isto é possível porque a avaliação ocorre usando-se grafos e não árvores.
- Veja o grafo que representa a avaliação da expressão acima e a arvore que acaba por duplicar o calculo de um elemento repetido. Os nós podem conter operadores e expressões.



(a) Duplicando 1+2 (b) sem duplicação de 1+2

Avaliação Preguiçosa

- Regras de cálculo usadas quando a avaliação se dá de uma forma preguiçosa.

1. Considerando funções definidas por padrões. Seja a função g definida a seguir:

```
g :: [Int] ->  [Int] -> Int

g [ ] y = 0 (1)

g (x:y) [ ] = 0 (2)

g (x:y) (w:z) = x+w (3)
```

Avaliação Preguiçosa

- Para determinar qual das equações (ou casos) deve ser usada, os argumentos devem ser calculados.
- Os argumentos não são avaliados completamente. Eles são avaliados até que se encontre um padrão que se case com eles.
- A busca ocorre de cima para baixo, isto é, da primeira em direção a ultima equação. Considerando a função g definida anteriormente, a avaliação de g [1..5] [1..5] geraria a seqüência de reduções:

```
g [1..5] [1..5]
 => g (1:[2..5]) [1..5] (uso do caso 2)
 => (1:[2..5]) (1:[2..5]) (uso do caso 3)
 => 1+1
 => 2
```

Avaliação Preguiçosa

- Regras de cálculo usadas quando a avaliação se dá de uma forma preguiçosa (cont.):

2. Considerando funções definidas por guardas:
   - Recapitulando, temos que os guardas são avaliados um de cada vez, de cima para baixo de acordo com a definição da função, até que um gere um valor True.
   - Uma vez que isto acontece, a expressão do lado direito é avaliada.

Avaliação Preguiçosa

- Para entender melhor como o cálculo é feito, a seguir são definidas uma função e uma expressão a ser calculada para ilustrar o processo de cálculo.

```
f :: Int -> Int -> Int -> Int

f x y z
    | x >= y && x >= z = x
    | y >= x && y >= z = y
    | otherwise = z
 =>  f (11+2) (3*4) (25-6)
```

Avaliação Preguiçosa

- A execução do programa acima geraria um resultado seguindo a ordem de cálculo descrita a seguir.

```
 f (11+2)(3*4) (25-6)

=> (11+2) >= (3*4) && (11+2) >= (25-6)

=> 13 >= 12 && 13 >= (25-6) => True && 13 >= (25-6)

=> 13 >= (25-6) => 13 >= 19

=> False -- o primeiro guarda está associado a um valor //
   False. Tenta-se agora o segundo guarda

=> 12 >= 13 && 12 >= 19 => False && 12 >= 19

=> False -- o segunda guarda também está associado a um //
   valor False.

=> otherwise = True

=>  19
```

Avaliação Preguiçosa

3. Considerando funções com definições locais: Os valores nas expressões where são calculados quando necessários. Veja o exemplo mostrado a seguir.

```
f :: Int  -> Int -> Int
f x y
   | naoVazia l = frente l
   | otherwise = y
     where
       l = [x..y]
```

Avaliação Preguiçosa

```
frente (x:(y:z)) = x+y

frente [x] = x

naoVazia [ ] = False

naoVazia (x:y) = True


=>  f 3 5
```

- A execução do programa acima implicará na seguinte seqüência de cálculo:

Avaliação Preguiçosa

- Execução:

```
f 3 5
naoVazia l
  | where
  | l = [3..5] => | l = (3: [4..5])
naoVazia (3:[4..5])
=>True => frente l
| where
   => l = (3:[4..5]) => l = (3:(4:[5]))
   => 3+4 => 7
```

Avaliação Preguiçosa

- Considerando a ordem quando existe uma escolha de funções a serem avaliadas. Seja o exemplo mostrado a seguir:

```
f :: Int -> Int

f a b = a+b

=>  f 1 f 3 4
```

Avaliação Preguiçosa

- Qual a ordem de cálculo que será seguida no calculo da expressão?
- Será o cálculo iniciado pela avaliação de f 3 4 ou começará pela avaliação de f 1 (f 3 4)?
- O cálculo da expressão inicia-se pela expressão mais externa. Dizemos que a avaliação da expressão ocorre de fora para dentro.