Listas em Haskell

Gustavo Zambonin
Paradigmas de Programação (UFSC-INE5416)

Nota: todos os excertos de código foram executados com `ghci ine5416_r9.hs` e chamados no interpretador.

Parte 1

- Classifica-se uma linguagem como *lazy*, ou "preguiçosa", quando esta age de modo a calcular uma expressão apenas quando necessário, não a cada vez que esta aparece ou é chamada em um código-fonte, evitando computações desnecessárias. Esta abordagem permite a potencial construção de estruturas infinitas de dados (por exemplo, `[1..]` em Haskell gerará uma lista equivalente a $\mathbb{N}^*$ até que não exista mais memória disponível para alocação). Em linguagens funcionais, é prático construir um programa que tenha um módulo gerador de possíveis respostas, e um módulo para selecionar uma resposta apropriada [pg. 9], assim necessitando apenas modificar o seletor de respostas, e nunca o gerador (que mesmo assim apenas computará o necessário).

- O uso de um mapeamento facilita a aplicação de certos critérios em um conjunto de elementos, como uma lista. As funções `map`, `reduce` e `filter` estão disponíveis por padrão em um interpretador para Python, e retornam novas listas, criadas a partir do critério (geralmente uma operação `lambda`, mas não se resumindo apenas a esta estratégia).

  - `list(map(chr, range(97, 123)))` retorna uma lista com os caracteres ASCII 97_{10} (`a` minúsculo) a 122_{10} (`z` minúsculo).
  - `list(map(lambda x: x**2, [1, 2, 3]))` retorna uma lista com os quadrados dos números na lista original.

- O módulo `Data.List` é composto de diversos utilitários para manipulação de listas, como um operador para concatenação destas, testes para lista vazia, retorno de tamanho de lista, funções para reversão, mapeamento, retorno de subsequências e permutações, entre outros.

Parte 2

- A implementação da soma de termos de uma progressão aritmética torna-se trivial quando Haskell oferece uma função `sum` que aceita listas como argumento. Desviando desta abordagem, implementa-se a fórmula genérica para soma de termos de uma PA da seguinte maneira: $S_n = \frac{n(2a_1+(n-1)r)}{2}$ é equivalente a

```
diff n = n!!1 - head(n)
sumAP n = (length n)*(2*head(n)+(length n - 1)*diff n) `div` 2
```

  Assume-se que a razão da PA é obtida por $a_2 - a_1$, como acontece na inferência do Haskell para construção de uma lista com tal característica.

- De modo similar, é possível obter o produto de todos os elementos de uma lista com a função `product`. Entretanto, desconsiderando esta solução prática, pode-se considerar a fórmula genérica para produto de termos de uma PA utilizando a função gama:

$$\Gamma(n) = (n-1)! = \int_0^\infty x^{n-1}e^{-x} \qquad \prod_{i=1}^{n} a_i = d^n \frac{\Gamma(a_1/d + n)}{\Gamma(a_1/d)}$$

  Utilizando a aproximação de Stirling para fatorial, calcula-se o valor do produto com a fórmula a seguir:

$$\Gamma(x) \approx \sqrt{\frac{2\pi}{x}} \left(\frac{1}{e} \left(x + \frac{1}{12x - 1/10x} \right) \right)^x$$

  É válido notar que, por conta de aproximação de ponto flutuante, a resposta torna-se discrepante à medida em que o número de termos da PA aumenta. A constante e é calculada a partir da série infinita $\sum_{n=0}^{\infty} \frac{1}{n!}$.

```
e = 1 + sum([1 / product[1..x] | x <- take 1000 [1..]])
gamma n = sqrt(2*pi / n) * (1/e * (n + (1/(12*n - 1/(10*n))))) ** n
productAP n = (diff n ** fromIntegral (length n)) *
    (gamma ((head(n) / diff n) + fromIntegral (length n)) /
    gamma (head(n) / diff n))
```