Universidade Federal de Uberlândia
Faculdade de Computação
Profª. Michele Nasu Tomiyama
Introdução à Computação 2 – Física

1. Recursão

A recursão é uma técnica bastante poderosa na programação funcional que permite que uma função seja definida através dela mesma. Ou seja, a função pode chamar a si mesma.

A idéia de qualquer algoritmo recursivo é simples: Se a instância em que estamos interessados é pequena, resolva-o diretamente, como puder. Se a instância é grande, reduza à uma instância menor do mesmo problema.

Uma função recursiva deve que seguir duas regras básicas:

- ter uma condição de parada – senão a função realizará computação infinita, ou seja, nuca irá parar.
- ter a chamada recursiva – onda a função chama a si mesma.

Exemplo1: Somar os n primeiros inteiros.

$$
\begin{array}{lcl}
 & & 1 + 2 + 3 + 4 + \ldots + n \\
soma\ 1 & = & \underbrace{1} \\
soma\ 2 & = & \underbrace{(soma\ 1) + 2} \\
soma\ 3 & = & \underbrace{(soma\ 2) + 3} \\
soma\ 4 & = & (soma\ 3) + 4 \\
 & \cdot & \\
 & \cdot & \\
 & \cdot & \\
soma\ n & = & (soma(n-1)) + n
\end{array}
$$

Sob uma definição matemática desta soma de inteiros entre 1 e n , é dada por:

$$
Soma(n) = \begin{cases} 1 & : n = 1 \\ Soma(n-1) + n & : n > 1 \end{cases}
$$

Neste caso, o código fica descrito como:

```
Soma :: Int -> Int
Soma n
     |n ==1 = 1
```

```
     |otherwise = soma (n-1) + n
```

Ou, sem as guardas:

```
Soma::Int->Int
Soma 1 = 1
Soma n = Soma (n-1) + n
```

Exemplo2: Série de Fibonacci

1, 1, 2, 3, 5, 8, 13, 21, 33,... onde:

- se n = 1 → fib 1 =1
- se n = 2 → fib 2 = 1
- se n > 2 → fib n = fib (n-1) + fib (n-2)

```
fib::Int->Int
fib n
     |n==1 || n==2 = 1
     |otherwise = fib(n-1) + fib(n-2)
```

2. Perigos da Recursão

Uma função recursiva é uma função que chama a si mesma, e quase todas as construções em Haskell são recursivas, pois necessitam de algum tipo de repetição. Praticamente em toda a função recursiva existe o conceito de aterramento (condição de parada), o qual deve sempre vir antes da linha que faz a chamada recursiva geral. Caso se inverta a sequência dessas linhas, uma sequência infinita de chamadas vai efetivamente ocorrer, se os critérios de parada não estiverem bem definidos. Uma função bem definida é aquela que possui um número de passos finitos, em que seus n aterramentos sempre se encontram antes da função recursiva geral.

Exemplo3:

```
soma::Int->Int
soma n = soma (n-1) + n
```

```
Main> soma 5
ERROR - C stack overflow
Main>
```

3. Avaliação Preguiçosa

A linguagem Haskell utiliza uma estratégia na avaliação das funções denominada avaliação preguiçosa (lazy evaluation), cujo fundamento é que não se avalia nenhuma subexpressão ou função até que o seu valor seja reconhecido como necessário.

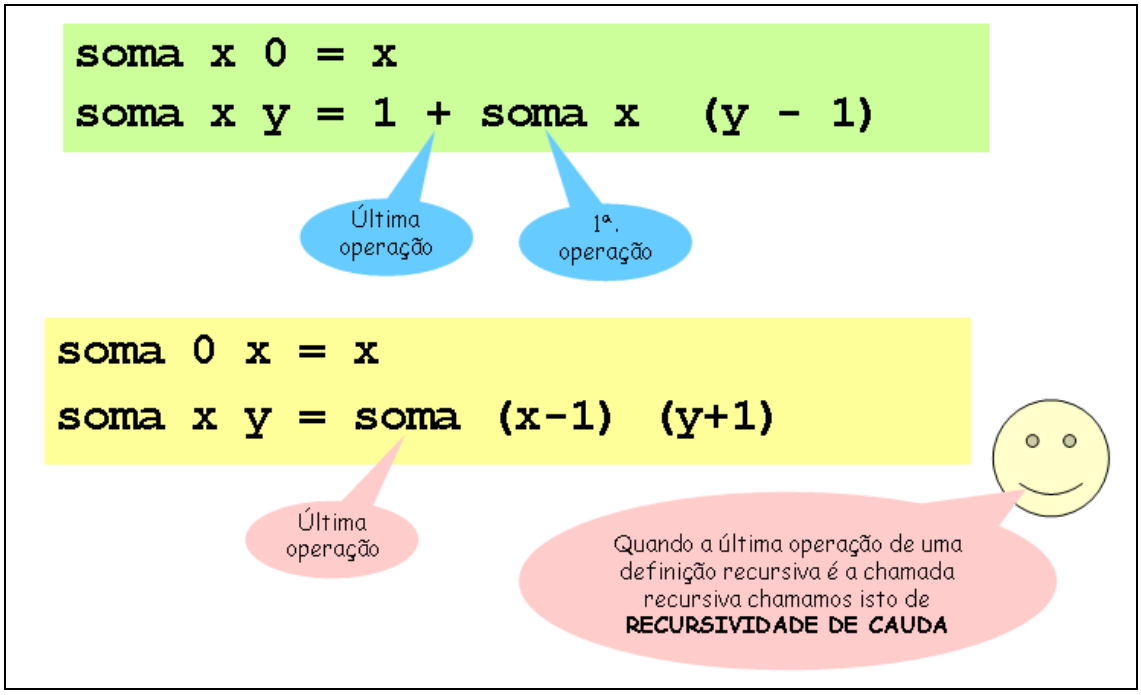
Exemplo:

Soma 7 = (Soma 6) + 7
= ((Soma 5) + 6) + 7

= (((Soma 4) + 5) + 6) + 7
= ((((Soma 3) + 4) + 5) + 6) +7
= (((((Soma 2) + 3) + 4) + 5) + 6) +7
= ((((((Soma 1) + 2) + 3) + 4) + 5) + 6) + 7
= ((((((1) + 2) + 3) + 4) + 5) + 6) + 7
= (((((1 + 2) + 3) + 4) + 5) + 6) + 7
= ((((3 + 3) + 4) + 5) + 6) + 7
= (((6 + 4) + 5) + 6) + 7
= ((10 + 5) + 6) + 7
= (15 + 6) + 7
= 21 + 7
= 28

4. Recursão de Cauda

A recursão de cauda é um tipo especial de recursão, no qual não existe processamento a ser feito depois de encerrada a chamada recursiva. Sendo assim, não é necessário guardar o estado do processamento no momento da chamada recursiva. Um compilador ou interpretador pode (se for construído assim) detectar a ocorrência de recursão de cauda e gerar código em que a chamada recursiva é implementada como um mero desvio do fluxo de instruções.



Exemplo4: Soma de 1 a n com recursão de cauda

```
somatorio::Int->Int
somatorio n = somaAux n 0

somaAux::Int->Int->Int
somaAux 0 x = x
somaAux x y = somaAux (x-1) (y+x)
```

```
somatório 5 = somaAux 5 0
            = somaAux 4 5
            = somaAux 3 9
            = somaAux 2 12
            = somaAux 1 14
            = somaAux 0 15
            = 15
```

5. Exercícios

1) Faça uma função que calcula o fatorial de um número.

2) Calcular a soma entre dois números n_1 e n_2 incluindo os limites
Soma 3 7 = 3+4+5+6+7 = 25

3) Seja a sequência:
$A_1 = \sqrt{6}$
$A_2 = \sqrt{6+\sqrt{6}}$
$A_3 = \sqrt{6+\sqrt{6+\sqrt{6}}}$
$A_4 = ...$
Encontre a forma recursiva para A_{n+1}

4) Faça uma função que calcula a potência X^Y, sem a utilização dos operadores de potenciação.

5) Faça funções utilizando recursão de cauda que calcula:
   a. O fatorial de um número.
   b. A potência X^Y.