DISCIPLINA: TEORIA DA COMPUTAÇÃO N
CÓDIGO: INF05501
SEMESTRE: 2018/1
TURMAS: A,B
PROFESSOR: Rafael Santos Coelho e Rodrigo Machado

DESCRIÇÃO DO TRABALHO PRÁTICO 2

PROGRAMAÇÃO EM CÁLCULO LAMBDA

Utilize o Simulador de Cálculo Lambda disponível em

http://www.inf.ufrgs.br/~rma/simuladores/lambda.html

para desenvolver as rotinas pedidas abaixo.

O trabalho consistirá em um único arquivo nomeado **trabalho.lam** , no qual os subitens de cada questão devem constar como **definições** no arquivo principal, **com os nomes definidos abaixo (os casos de teste usarão os nomes mencionados – não altere os mesmos e preste atenção em maiúsculas e minúsculas). A expressão principal do programa não será considerada na correção.**

Envie (via Moodle) um arquivo .ZIP contendo o arquivo do programa desenvolvido, junto com um arquivo de texto indicando os componentes do grupo. Somente um componente do grupo deverá fazer a submissão (pelo grupo inteiro).

EXERCÍCIOS

1. Implemente as seguinte operações em Cálculo Lambda. Nessa questão, utilize numerais de Church para representar números naturais. Quando for solicitado representar funções de mais de um argumento, faça funções de alta ordem que recebem uma argumento de cada vez (currying).

a) f1 x y = 2*x + y (NOME: `f1`)

b) f2 (a,b) = (b,a) // swap de pares (NOME: `f2`)

c) f3 [x1,x2,…,xn] = x1 + x2 + … + xn // soma todos os números da lista
// retorna 0 se lista vazia (NOME: `f3`)

d) f4 [x1, x2, … , xn] = [xn, … , x2, x1] // devolve o reverso da lista de entrada (NOME: `f4`)

e) f5 [x1,x2,...xn] [y1,y2,…,yn] = [(x1,y1), (x2,y2), … , (xn,yn)]
// Recebe duas listas, e cria lista de pares de elementos.
// Se as listas de entrada forem de tamanho distinto
// a lista resultante deve possuir o tamanho da menor lista
// de entrada (NOME: `f5`)

2. Os numerais de Church não são a única forma de representar números naturais em Cálculo Lambda. Por exemplo, a seguir será apresentada a chamada *codificação padrão* (que faz uso dos termos **pair** e **false** vistos na disciplina).

```
0    = \x.x
succ = \n. (pair false n)
```

Nessa representação, o número 0 é representado pela função identidade, e um número n é representado por um encadeamento de pares ordenados contendo sempre false como primeiro componente.

```
Ex:   3 = (pair false (pair false (pair false (\x.x) ) ) )
      5 = (pair false (pair false (pair false (pair false (pair false (\x.x) ) ) ) ) )
```

Na codificação padrão, podemos implementar as operações predecessor e teste de zero como segue:

```
isZero = fst
pred   = snd
```

Note que a operação de predecessor é significativamente mais eficiente que a mesma operação utilizando numerais de Church. Em contrapartida, não é possível aplicar um número como função diretamente para obter n repetições de uma operação sobre um argumento inicial.

Implemente as seguintes rotinas utilizando números naturais representados via *codificação padrão:*

a) repeatCP n f x = (f (f (f … (f x) …))) (n vezes) (NOME: `repeatCP`)

b) evenCP x = true se x for um número par, false caso contrário (NOME: `evenCP`)

c) addCP x y = x+y (NOME: `addCP`)

d) multCP x y = x*y (NOME: `multCP`)

e) halfCP x = ⌊x/2⌋ (NOME: `halfCP`)