Lista de Linguagens de Programação – 9

Nome: ______________________________ Matrícula: _______________

1. Nós podemos representar números inteiros usando o cálculo λ. Uma das convenções mais comuns é assumir que um número n é uma função que recebe dois argumentos, e aplica o primeiro ao segundo n vezes. Por exemplo:

   - $0 = \lambda s.\lambda z.z$
   - $1 = \lambda s.\lambda z.sz$
   - $2 = \lambda s.\lambda z.s(sz)$

   Podemos também representar valores booleanos usando o cálculo λ. Por exemplo:

   - $F = \lambda x.\lambda y.y$
   - $T = \lambda x.\lambda y.x$

   (a) Considere a função $\text{MUL} = \lambda n_1.\lambda n_2.\lambda z.n_1(n_2\ z)$. Usando a definição do número 2 acima, mostre todos os passos da redução MUL 2 2.

   (b) Usando a função sucessor, $\text{SUCC} = \lambda n.\lambda y.\lambda x.y(n\ y\ x)$, defina a função ADD, que soma dois números.

   (c) Defina uma função XOR, que receba dois valores booleanos b_1 e b_2, definidos como convencionado acima, e retorne T caso $b_1 \neq b_2$ e F caso contrário.

2. Encontre o conjunto de variáveis livres para as seguintes expressões λ:

   (a) $\lambda x.xy\lambda z.xz$

   (b) $(\lambda x.xy)\lambda z.w\lambda w.wzyx$

   (c) $x\lambda z.x\lambda w.wzy$

   (d) $\lambda x.xy\lambda x.yx$

3. Podemos usar a notação $e[x \to y]$ para denotar a aplicação $(\lambda x.e)y$. Execute as substituições abaixo:

   (a) $(f(\lambda x.xy)\lambda z.xyz)[x \to g]$

   (b) $((\lambda x.fx)\lambda f.fx)[f \to gx]$

   (c) $(\lambda x.\lambda y.fxy)[y \to x]$

   (d) $(\lambda f.\lambda y.fxy)[x \to fy]$

4. Essa questão diz respeito as regras de precedência usadas para construir e interpretar expressões λ.

    (a) Qual a diferença entre $\lambda x.(xy)$ e $(\lambda x.x)x$? Pelas nossas convenções de precedência e associatividade, qual daquelas duas expressões é equivalente a $\lambda x.x\ y$?

    (b) A gramática abaixo descreve as expressões λ de forma não ambígua. Segundo essa gramática, qual deveria ser a intepretação de $\lambda x.x\ y$? Justifique a sua resposta exibindo uma árvore de derivação.

$$
\begin{array}{rcl}
< expr > & ::= & < atom > \ | \ < app > \ | \ < fun > \\
< atom > & ::= & < head > \ | \ < app > \\
< head > & ::= & < var > \ | \ (< fun >) \\
< app > & ::= & < head >< atom > \ | \ < app >< atom > \\
< fun > & ::= & \lambda < var > . < expr >
\end{array}
$$

5. Uma expressão λ como $(\lambda x.x)y$ pode ser reduzida, via uma redução β, em y. A expressão y já não pode mais ser reduzida. Se uma expressão não pode mais sofrer qualquer redução, dizemos que aquela expressão está na **forma normal**.

    (a) Nem toda expressão λ possui uma forma normal. Escreva uma expressão para a qual a forma normal não existe.

    (b) O fato de existirem expressões que não possuem forma normal é essencial para que o cálculo λ seja computacionalmente equivalente à Máquina de Turing. Por que?

6. Considere as expressões $S = \lambda xyz.(xz)(yz)$ e $K = \lambda xy.x$. Qual a forma normal de $S\ K\ K$? Note que esse exercício aparentemente simples pode ficar complicado se você não aplicar as reduções com cuidado. *Dica:* use as abreviações S e K tanto quanto possível; isto é, faça as substituições com termos λ apenas quando você realmente precisar.

7. Suponha que um único símbolo no cálculo λ possua 5 milímetros de largura. Escreva uma expressão λ com no máximo 20 centímetros de largura cuja forma normal possua pelo menos $10^{10^{10}}$ anos-luz de comprimento.

8. Uma linguagem que possui: (i) chamadas recursivas de função; (ii) o valor zero; (iii) a função predecessor; (iv) a função successor e (v) a função `zero?`, que testa se um número é zero, é Turing Completa. O nosso cálculo λ é Turing Completo, logo é possível escrever a função `zero?` nessa linguagem. Escreva tal função. Lembre-se: números possuem a forma $\lambda f.\lambda x.f(\ldots(fx))$, de forma tal que o número n representa n aplicações da função f sobre o parâmetro x.

9. Esta questão refere-se ao programa SML abaixo:

```
fun insertHead _ nil = nil
  | insertHead e (h::t) = (e::h) :: insertHead e t

fun comb 0 _ = [[]]
  | comb _ [] = []
  | comb n (h::t) = insertHead h (comb (n-1) t) @ comb n t
```

(a) Qual o tipo da função `insertHead`?

(b) Qual o resultado da chamada `insertHead 3 [[1], [2]]`?

(c) Qual o tipo da função `comb`?

(d) Qual o resultado da chamada `comb 2 [1,2,3]`?

(e) Qual a complexidade assimptótica da função `comb`?