PROVA 2 DE TEORIA

Bento Borges Schirmer 16/1/23

1. a) Apenas as afirmativas I e III

2. Essa definição não verifica as variáveis livres, e por isso está sujeita a capturar ou liberar variáveis durante a substituição

$$[x \mapsto y](\lambda x.x) = \lambda x.y$$

variável ligada

variável liberada, não deveria

3.

$$\begin{tikzcd}
(\lambda x.x)((\lambda x.x)(\lambda z.(\lambda x.x)z)) \arrow[d, "\beta"] \arrow[ddd, bend right, "\beta"'] \\
(\lambda x.x)(\lambda z.(\lambda x.x)z) \arrow[d, "\beta"] \\
\lambda z.(\lambda x.x)z \\
(\lambda x.x)(\lambda z.(\lambda x.x)z) \arrow[d, "\beta"] \\
(\lambda z.(\lambda x.x)z)
\end{tikzcd}$$

call-by-name

call-by-value

A diferença pode ser ilustrada com as estratégias usadas em C e Haskell: em C, os parâmetros são avaliados primeiro e os valores devolvidos são passados prontos pro corpo da função (call-by-value); em Haskell, a função é executada primeiro e os parâmetros avaliados conforme necessário, posteriormente.

4.

```
#include <stdbool.h>

struct term
{
    enum {TOKEN, ABS, APP} type;

    char t;
    struct term *t1;
    struct term *t2;
};

bool has_lambda(struct ~~token~~ term t[static 1])
{
    switch (t->type)
    {
        case TOKEN: return false;
        case ABS  : return true;
        case APP  : return has_lambda(t->t1) || has_lambda(t->t2);
    }
}

bool ~~h~~ contains_beta_redex(struct term t[static 1])
{
    switch (t->type)
    {
        case TOKEN: return false;
        case ABS  : return contains_beta_redex (t->t1);
        case APP  :
            return has_lambda(t->t1) || contains_beta_redex(t->t2);
    }
}
```