

# 198:515 Programming Languages and Compilers I

## Problem Set 3

This homework will not be graded. A sample solution will be made available at a later time.

### Problem 1 - $\lambda$ -term abbreviations

Give the fully expanded  $\lambda$ -terms for the following  $\lambda$ -term abbreviations as discussed in class (lecture 6, page 15).

1.  $(f\ g\ x\ v)\ w$
2.  $\lambda xy.(\lambda z.x(yz))v\ w$
3.  $\lambda x.xx\ \lambda x.xx$

### Problem 2 - $\beta$ -reductions

Show all possible  $\beta$ -reduction sequences for the  $\lambda$ -term

$$(\lambda xyz.(xz)(yz))\ (\lambda xy.x)\ (\lambda xy.x)$$

Clearly mark the redex at each step (see lecture 6, page 17)

### Problem 3 - programming in lambda calculus

Logical constants and operations can be represented in lambda calculus as follows:

$$\begin{aligned} \mathbf{true} &\equiv \lambda a.\lambda b.a && \textit{select-first} \\ \mathbf{false} &\equiv \lambda a.\lambda b.b && \textit{select-second} \\ \mathbf{not} &\equiv \lambda x.((x\ \mathbf{false})\ \mathbf{true}) \\ \mathbf{and} &\equiv \lambda x.\lambda y.((x\ y)\ \mathbf{false}) \\ \mathbf{or} &\equiv \lambda x.\lambda y. ((x\ \mathbf{true})\ y) \end{aligned}$$

Give the lambda calculus implementation of the boolean operations:

1. **implication**  $(a \rightarrow b)$
2. **exclusive or**  $(a\ \text{exor}\ b)$

### Problem 4 - reductions

1. Give three  $\lambda$ -terms that do not have a normal form

2. Give two  $\lambda$ -terms that reduce to normal form under left-most redex and call-by-name, but **not** left-most redex and call-by-value
3. Give a  $\lambda$ -term for which the normal form and the head-normal form are distinct.

## Problem 5 - Scheme programming

Write the following functions on lists in Scheme. The semantics of the functions is described through examples.

1.
 

```
(define remove-all
  (lambda (l)
    ...))
...
(remove-all '(a((b)(c d)((e f)))))) --> '((( )((( )))
```
2.
 

```
(define reverse-list
  (lambda (l)
    ...))
...
(reverse-list '(a((b)(c d)((e f g)))))) --> '((((g f e))(d c)(b))a)
```
3.
 

```
(define make-single
  (lambda (l)
    ...))
...
(make-single '(a((b)(c c)((e e e)))))) --> (a((b)(c)((e))))
```

## Problem 6 - TINY interpreter

Modify the static scoping simulator available on the ilab cluster

`~uli/cs515/examples/scheme/ValueStatic.ss`

to implement dynamic scoping instead of static scoping.