

198:515 Programming Languages and Compilers I
Fall 2007, Problem Set 5
Sample Solution

Problem 1

1. Show all possible β -reduction sequences for the λ -term.
 Clearly mark the redex at each step (see lecture 12, page 16)
 ANSWER: Please see last page.
2. Sethi, page 576: 14.2 a)

lambda = \
 part a

Prove $SIII = \{\beta\text{-reductions}\} I$ where $S \Rightarrow \lambda xyz.(xz)(yz)$ and $I \Rightarrow \lambda x.x$

now for clarity we can write :-

SIII as
 $(\lambda xyz.(xz)(yz))(\lambda x'.x')(\lambda x''.x'')$ \Rightarrow b-reduction starting from left to
right
 $\Rightarrow ((\lambda xyz.(xz)(yz))(\lambda x'.x'))(\lambda x''.x'')(\lambda x'''.x''')$
 $\Rightarrow (\lambda yz.((\lambda x'.x')z)(yz))(\lambda x''.x'')(\lambda x'''.x''')$
 $\Rightarrow ((\lambda z.((\lambda x'.x')z)((\lambda x''.x'')z))(\lambda x'''.x'''))$
 $\Rightarrow ((\lambda x'.x')(\lambda x'''.x'''))((\lambda x''.x'')(\lambda x'''.x'''))$
 $\Rightarrow (\lambda x'''.x''')((\lambda x''.x'')(\lambda x'''.x'''))$
 $\Rightarrow \{((\lambda x''.x'')(\lambda x'''.x'''))/x'''\}(\lambda x'''.x''') \Rightarrow ((\lambda x''.x'')(\lambda x'''.x'''))$
 $\Rightarrow \{x'''/x'''\}(\lambda x''.x'') \Rightarrow \lambda x'''.x''' = I$

Problem 2

On page 28. lecture 6, we talked about representing lists as lambda terms. Give a lambda term that represents the function *isEmpty?*. This term should return either the lambda term for true or false. Hint: Assume that the empty list is represented as:

`empty` $\equiv \lambda x. \text{true}$

Answer: `isEmpty?` $\equiv \lambda x. (x (\lambda y. \lambda z. \text{false}))$

Problem 3

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

1.

```
(define replace
  (lambda (atom1 atom2 l)
    ...))
...
(replace 'c 'd '(a((b)(c)((e e)))))) --> (a((b)(d)((e e))))
```

POSSIBLE SOLUTION:

```
(define (replace a b l)
  (cond ((null? l) l) ;; if l is empty
        ((not (pair? l)) (if (eq? a l) b l)) ;; if l is a singleton
        ;; if l is a list
        (else (cons (replace a b (car l)) (replace a b (cdr l))))))
```

2.

```
(define double-flatten
  (lambda (l)
    ...))
...
(double-flatten '(a((b)(c d)((e)))))) --> '(a a b b c c d d e e)
```

POSSIBLE SOLUTION:

```
(define (double-flatten l)
  (cond ((null? l) l) ;; if l is empty
        ((not (pair? l)) (list l l)) ;; if l is a singleton
        ;; if l is a list
        (else (append (double-flatten (car l)) (flatten-double (cdr l))))))
```

3.

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

POSSIBLE SOLUTION:

```
(define (make-single l)
  (cond ((null? l) l) ;; if l is empty
        ((not (pair? l)) l) ;; if l is a singleton
```

```
;; if l is a list (two cases)
;; (case 1) duplicates
((and (not (null? (cdr l))) (eq? (car l) (cadr l)))
 (make-single (cdr l)))
;; (case 2) no duplicates
(else (cons (make-single (car l)) (make-single (cdr l))))))
```