Solutions to Sample Final Exam Questions

Problem 1

\[ ((a \ b \ c) \ a \ b \ c) \]

\[ ((a) \ a) \]

\[ ((a \ a) \ a) \]
**Problem 2**

\[
\begin{align*}
\text{(define (map-two-streams f s1 s2) } & \text{ (cons-stream (f (stream-car s1)} \\
& \text{ (stream-car s2))) } \text{ (map-two-streams f } \\
& \text{ (stream-cdr s1)} \\
& \text{ (stream-cdr s2)))})
\end{align*}
\]

There is one multiplication done to compute each element of the stream, so the nth element will need n multiplications. However, if the answer was already computed, no multiplications would be necessary. Or if the (n-1)th element had been computed, just one more multiplication is necessary.

Without memoization: The nth element still needs n multiplications to be computed. However, if we ask for the (n+1)th element after computing n, we’ll need to redo all of the prior n multiplications since we’re not using memoization.

**Problem 3**

\[
\begin{align*}
\text{(define (make-opl-lecturer name birthplace threshold) } & \text{ (let ((person (make-person name birthplace threshold))) } \\
& \text{ (lambda (message) } \text{(cond ((eq? message 'bring-cookies-to-exam) } \\
& \text{ (lambda (self) } \text{(let ((location (ask self 'place))) } \\
& \text{ (ask self 'move-to cookie-store) } \\
& \text{ (ask self 'take cookies) } \\
& \text{ (ask self 'move-to location)))}) \\
& \text{ ((eq? message 'say) } \\
& \text{ (lambda (self stuff) } \text{(ask person 'say stuff) } \\
& \text{ (ask person 'say '(Scheme is fun!)))}) \\
& \text{ (else (get-method person message)))})}
\end{align*}
\]
Problem 4

Insert before application? in mc-eval:

```scheme
((infix? exp)
  (mc-apply (mc-eval (infix-operation exp) env)
             (list-of-values (infix-operands exp) env)))
```

Rest of necessary code:

```scheme
(define (infix? exp)
  (or (eq? (cadr exp) 'uml:+)
      (eq? (cadr exp) 'uml:-)
      (eq? (cadr exp) 'uml:*)
      (eq? (cadr exp) 'uml:/)))

(define (infix-operator exp) (cadr exp))

(define (infix-operands exp)
  (cons (car exp) (cddr exp)))
```

Problem 5

```scheme
(define (make-frame variables values)
  (if (null? variables)
      nil
      (cons (cons (car variables) (car values))
            (make-frame (cdr variables) (cdr values)))))
```

Here’s the complete lookup-variable-value function. The underlined portions show what was changed from the original.

```scheme
(define (lookup-variable-value var env)
  (define (env-loop env)
    (define (scan frame)
      (cond ((null? frame)
              (env-loop (enclosing-environment env)))
            ((eq? var (caar frame)) (cdar frame))
            (else (scan (cdr frame))))
    (if (eq? env the-empty-environment)
        (error "Unbound variable" var)
        (scan (first-frame env))))
  (env-loop env))
```

The following procedures would need to be changed:

- set-variable-value!
- define-variable!
- add-binding-to-frame!
- frame-variables
- frame-values