Problem Set 2

Due: 25 September 1995.

Reading for this problem set: Handout 2, "A Term Rewriting Model for Scheme, I."

Problem 1. Consider the following adaptations to inequalities of the two substitution rules for equality:

$$(1) \qquad \models f \leq g \text{ implies } \models e[x := f] \leq e[x := g].$$

$$(2) \qquad \models e \leq g \text{ implies } \models e[x := f] \leq g[x := f].$$

Only one of these assertions is true. Prove the one which is true, and give a simple counterexample to the other one. You may assume from lecture the

Lemma. $[\![e[x := f]]\!]V = [\![e]\!](V[x \leftarrow [\![f]\!]V])$,

where for any function, F, and elements a, b, we define $F[a \leftarrow b]$ to be the function G such that

$$G(u) = F(u) \text{ for } u \neq a, \text{ and } G(a) = b.$$

Problem 2. For each numbered step in the sample rewriting of `(iter-factorial 3)` attached at the end of this Problem Set, indicate the subexpression which gets rewritten and the rules applied ("P"–primitive application, "C"–conditional, "I"–instantiation, "L"–lambda application, "G"–garbage collection)

Problem 3. Inspection of the successive steps in the evaluation of `(iter-factorial 3)` reveals the order in which the rewriting interpreter evaluates the subexpressions of a combination, e.g, left-to-right or right-to-left. What is the order? Explain how you reached this conclusion, indicating the particular steps of evaluation which reveal the order.

Problem 4. Let A be any one of the Scheme term-rewriting rules P, C, I, L, G. We use the notation $M \underset{\text{A}}{\longrightarrow} N$ to indicate that scheme body M rewrites to become body N by a *single* application of rule A, and $M \underset{\text{A}*}{\longrightarrow} N$ to indicate any number, including zero, applications of rule A.

4(a) Show that if $M \underset{\text{A}}{\longrightarrow} M_1$ and also $M \underset{\text{G}}{\longrightarrow} M_2$, then there is an M_3 such that $M_1 \underset{\text{G}*}{\longrightarrow} M_3$ and either $M_2 = M_3$ or $M_2 \underset{\text{A}}{\longrightarrow} M_3$.

4(b) It now follows that it doesn't matter when garbage collections are performed in the course of rewriting a Scheme expression into a value. Give a precise statement of this fact (along the lines of the Uniqueness Theorem of the handout "A Term Rewriting Model for Scheme"), and outline its proof.

Rewriting Evaluation of Iterative Factorial

```
(smeval
'((define (iter n result)
  (if (<= n 0)
      result
      (iter (- n 1) (* n result))))
(define (iter-factorial n)
  (iter n 1))
(iter-factorial 3)))

;==(0)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (define iter-factorial
   (lambda (n)
     (iter n 1)))
 (iter-factorial 3))

;==(1)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 ((lambda (n) (iter n 1)) 3))

;==(2)==>
((define iter (lambda (n result) (if (<= n 0) result (iter (- n 1) (* n result)))))
 (define n_21 3)
 (iter n_21 1))

;==(3)==>
((define iter (lambda (n result) (if (<= n 0) result (iter (- n 1) (* n result)))))
 (iter 3 1))

;==(4)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 ((lambda (n result) (if (<= n 0) result (iter (- n 1) (* n result)))) 3 1))

;==(5)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (define n_22
   3)
 (define result_17
   1)
 (if (<= n_22 0)
     result_17
     (iter (- n_22 1) (* n_22 result_17))))

;==(6)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
```

```
 (define n_22
   3)
 (define result_17
   1)
 (if (<= 3 0)
     result_17
     (iter (- n_22 1) (* n_22 result_17))))

;==(7)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (define n_22
   3)
 (define result_17
   1)
 (if #f
     result_17
     (iter (- n_22 1) (* n_22 result_17))))

;==(8)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (define n_22
   3)
 (define result_17
   1)
 (iter (- n_22 1) (* n_22 result_17)))

;==(9)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (define n_22
   3)
 (iter (- n_22 1) (* n_22 1)))

;==(10)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (define n_22
   3)
 (iter (- n_22 1) (* 3 1)))

;==(11)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (define n_22
   3)
 (iter (- n_22 1) 3))

;==(12)==>
((define iter
   (lambda (n result)
```

```
      (if (<= n 0)
          result
          (iter (- n 1) (* n result)))))
 (iter (- 3 1) 3))

;==(13)==>
((define iter (lambda (n result) (if (<= n 0) result (iter (- n 1) (* n result)))))
 (iter 2 3))

;==(14)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 ((lambda (n result) (if (<= n 0) result (iter (- n 1) (* n result)))) 2 3))

;==(15)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (define n_23
   2)
 (define result_18
   3)
 (if (<= n_23 0)
     result_18
     (iter (- n_23 1) (* n_23 result_18))))

;==(16)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (define n_23
   2)
 (define result_18
   3)
 (if (<= 2 0)
     result_18
     (iter (- n_23 1) (* n_23 result_18))))

;==(17)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (define n_23
   2)
 (define result_18
   3)
 (if #f
     result_18
     (iter (- n_23 1) (* n_23 result_18))))

;==(18)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (define n_23
   2)
```

```
 (define result_18
   3)
 (iter (- n_23 1) (* n_23 result_18)))

;==(19)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (define n_23
   2)
 (iter (- n_23 1) (* n_23 3)))

;==(20)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (define n_23
   2)
 (iter (- n_23 1) (* 2 3)))

;==(21)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (define n_23
   2)
 (iter (- n_23 1) 6))

;==(22)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (iter (- 2 1) 6))

;==(23)==>
((define iter (lambda (n result) (if (<= n 0) result (iter (- n 1) (* n result)))))
 (iter 1 6))

;==(24)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 ((lambda (n result) (if (<= n 0) result (iter (- n 1) (* n result)))) 1 6))

;==(25)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (define n_24
   1)
 (define result_19
   6)
 (if (<= n_24 0)
     result_19
     (iter (- n_24 1) (* n_24 result_19))))
```

```
;==(26)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (define n_24
   1)
 (define result_19
   6)
 (if (<= 1 0)
     result_19
     (iter (- n_24 1) (* n_24 result_19))))

;==(27)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (define n_24
   1)
 (define result_19
   6)
 (if #f
     result_19
     (iter (- n_24 1) (* n_24 result_19))))

;==(28)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (define n_24
   1)
 (define result_19
   6)
 (iter (- n_24 1) (* n_24 result_19)))

;==(29)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (define n_24
   1)
 (iter (- n_24 1) (* n_24 6)))

;==(30)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (define n_24
   1)
 (iter (- n_24 1) (* 1 6)))

;==(31)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result))))))
```

```
 (define n_24
   1)
 (iter (- n_24 1) 6))

;==(32)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (iter (- 1 1) 6))

;==(33)==>
((define iter (lambda (n result) (if (<= n 0) result (iter (- n 1) (* n result)))))
 (iter 0 6))

;==(34)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 ((lambda (n result) (if (<= n 0) result (iter (- n 1) (* n result)))) 0 6))

;==(35)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (define n_25
   0)
 (define result_20
   6)
 (if (<= n_25 0)
     result_20
     (iter (- n_25 1) (* n_25 result_20))))

;==(36)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (define n_25
   0)
 (define result_20
   6)
 (if (<= 0 0)
     result_20
     (iter (- n_25 1) (* n_25 result_20))))

;==(37)==>
((define iter
   (lambda (n result)
     (if (<= n 0)
         result
         (iter (- n 1) (* n result)))))
 (define n_25
   0)
 (define result_20
   6)
 (if #t
     result_20
     (iter (- n_25 1) (* n_25 result_20))))

;==(38)==>
```

```
((define result_20 6) result_20)

;==(39)==>
6
;Value 1: "Syntactic Value was returned"
```