Problem Set 3

Due: 4 October, 1996.

Reminder: **QUIZ 1** in class on 4 October.

Reading assignment. Revised (27 September) version of Handout 2, "A Term Rewriting Model for Scheme, I."

Problem 1. We consider applying the garbage-collection rule to the body

```
(define u (lambda () (* 2 x)))
(define v 4)
(define w (+ 1 (u)))
(define x (+ 1 v))
(define y (lambda () x))
(define z (lambda () y))
(cons v x)
```

For each of the subproblems below, briefly explain how you arrived at your answer.

1(a) Indicate which definitions can be garbage-collected "immediately", that is, by one application of the garbage-collection rule to this body.

1(b) Indicate which of the above definitions get garbage-collected when this body is rewritten to a value while obeying the Lambda Body Restriction.

1(c) Indicate which of the above definitions can be garbage-collected when this body is rewritten to a value with unrestricted application of rewrite rules.

Problem 2. We define *simple expressions* to be those expressions of kernel Scheme (cf. Handout 2, Appendix A) which do not contain lambda expressions or ⟨symbol⟩'s, and whose *only* occurrences of variables are the primitive procedure variables `+`, `-`, `*`, `<`, and `=`. (So simple expressions can't contain ⟨define⟩'s either, since procedure variables can't be redefined in our TRM). We also require that if one of these procedure variables occurs as the operator of a combination, then the combination has exactly two operands.

2(a) Give a grammatical description of ⟨simple expression⟩'s in the style of Appendix A of Handout 2.

The set of *simple values* is defined to be the integers and truth values along with three functions from pairs of integers to integers, namely, addition, subtraction and multiplication, and also with two binary relations on the integers, namely, equality and $<$. (Warning: A simple value is an abstract *mathematical* object. A ⟨primitive value⟩ or a ⟨procedure variable⟩ is a *syntactic* token. Don't confuse them.)

We regard any simple expression, s, as describing some simple value which we call the *meaning of* s, written $[\![s]\!]$, except that not every ⟨simple expression⟩ is meaningful. For example, `(+ #t <)` is meaningless because it makes no sense to add a truth-value to a relation. Similarly, `(- 7 (+ #t <))` is meaningless because it makes no sense to subtract something meaningless from the integer seven. On the other hand, it makes sense to declare $[\![$`(if #f (- 7 (+ #t <)) =)`$]\!]$ to be the same as $[\![$`=`$]\!]$, namely, the integer equality relation, even though the first branch of this conditional is meaningless.

2(b) We can regard "meaning" as a partial function, $[\![\cdot]\!]$, from simple expressions to simple values. Give a definition of this partial function in the style used in Handout 3 to define the meaning of arithmetic expressions—namely, by structural induction on ⟨simple expression⟩'s.

2(c) Prove that if s_1 and s_2 are simple expressions and $s_1 \longrightarrow s_2$, then s_2 is a simple expression, and either both of s_1 and s_2 are meaningless, or else $[\![s_1]\!] = [\![s_2]\!]$.

2(d) Prove the following version of a Unique Value Theorem for Simple Expressions:

Theorem. If a simple expression is meaningful, then it rewrites to a *unique* ⟨immediate value⟩.

Hint: Use structural induction on ⟨simple expression⟩'s. You will need a stronger induction hypothesis than this theorem itself. It will help to note that for every simple value, v, there is a unique simple expression, call it $\hat{v}$, which is an ⟨immediate value⟩ such that $[\![\hat{v}]\!] = v$.

2(e) Prove that every simple expression, meaningful or not, rewrites to a *unique, terminated* simple expression.

Problem 3. Two Scheme expressions or bodies are said to be *equivalent up to renaming* iff they can be made syntactically identical by freshly renaming their bound variables. Equivalence up to renaming is often called α-*equivalence* in the research literature and is written $=_\alpha$.

3(a) Describe (in English or in Scheme) a procedure *replace* such that for any Scheme body, B, and variables x, y, applying *replace* to the arguments B, x, y yields B with all *free* occurrences of x replaced by y.

3(b) Describe (in English or in Scheme) a procedure which determines whether two Scheme bodies are $=_\alpha$.

Hint: Suppose B_1 is a body of the form `(lambda (a b)` C_1`)`. Then $B_2 =_\alpha B_1$ iff B_2 is of the form `(lambda (c d)` C_2`)`, and there are variables $v_1 \neq v_2$ which do not occur in either of B_1 or B_2 such that

$$\mathit{replace}(\mathit{replace}(C_1, \texttt{a}, v_1), \texttt{b}, v_2) =_\alpha \mathit{replace}(\mathit{replace}(C_2, \texttt{c}, v_1), \texttt{d}, v_2).$$

3(c) *optional:* Prove that $=_\alpha$ is an equivalence relation on Scheme bodies.

Problem 4. Suppose B_1, B_2 are Scheme bodies such that $B_1 =_\alpha B_2$.

4(a) Suppose $B_1 \longrightarrow C_1$. Verify that there must be a $C_2 =_\alpha C_1$ such that $B_2 \longrightarrow C_2$.

4(b) Suppose B_1 can be rewritten to the numeral `13` using our Scheme Term Rewriting Model (TRM). Conclude that B_2 also can be rewritten to `13`.

Problem 5. Show that the Unique Primitive Value Theorem would fail if we tried to specify the Scheme procedure `equal?` by adding the following rewrite rules to our TRM for functional Scheme:

`(equal?` ⟨immediate value$_1$⟩ ⟨immediate value$_2$⟩`)` $\longrightarrow$ `#t`,
if ⟨immediate value$_1$⟩ and ⟨immediate value$_2$⟩ are $=_\alpha$.

`(equal?` ⟨immediate value$_1$⟩ ⟨immediate value$_2$⟩`)` $\longrightarrow$ `#f`,
if ⟨immediate value$_1$⟩ and ⟨immediate value$_2$⟩ are not $=_\alpha$.