Problem Set 3

Due: 13 October 1995.

Problem 1. The `Y` operator described in class satisfied the property that

$$\texttt{(Y } V\texttt{) = (}V\texttt{ (lambda (z) ((Y } V\texttt{) z)))}$$

for any immediate value V. This allows us to simulate recursive definitions of one argument procedures, i.e., of the form `(define (f x)` ⟨expr⟩) by letting $V =$ `(lambda (f) (lambda (x)` ⟨expr⟩)).

1(a) Describe a similar operator `Y2` such that that `(Y2` V`) = (`V `(lambda (u v) ((Y2` V`) u v)))` for any immediate value V, so `Y2` can be used to simulate recursive definitions of two-argument procedures `(define (g x y)` ⟨expr⟩).

1(b) Explain what happens if we used `Y`, instead of `Y2`, on `g` above? (You might like to try this using the `smeval` procedure.)

1(c) (Optional, for discussion) Is there a `Y`-like operator which handles

1. `(define x (lambda () x))`?
2. How about `(define x 3)`?
3. `(define x x)`?
4. `(define x (* 0 x))`?
5. `(define x (cons 0 x))`?

Problem 2. We use the notation $M \underset{\text{rew}*}{\longrightarrow} N$ to indicate that expression M can be rewritten to be N by zero or more applications of rules of the Scheme term rewriting model.

A set, $\mathcal{P}$, of Scheme expressions is *evaluation-invariant* if, whenever $M \underset{\text{rew}*}{\longrightarrow} N$, then $M \in \mathcal{P}$ iff $N \in \mathcal{P}$.

Two sets $\mathcal{P}, \mathcal{Q}$ of expressions are *Scheme-separable* iff there is a Scheme procedure, `good-side`, such that for every expression ⟨expr⟩,

1. there is a truth-value b such that `(good-side '`⟨expr⟩`)` $\underset{\text{rew}*}{\longrightarrow} b$,
2. `(good-side '`⟨expr⟩`)` $\underset{\text{rew}*}{\longrightarrow}$ `#t` whenever ⟨expr⟩ $\in \mathcal{P}$, and
3. `(good-side '`⟨expr⟩`)` $\underset{\text{rew}*}{\longrightarrow}$ `#f` whenever ⟨expr⟩ $\in \mathcal{Q}$.

2(a) Show that if P, Q are nonempty, evaluation-invariant sets of expressions, then they are not Scheme-separable. *Hint*: Consider

```
(define (contradict expr)
    (define (make-quoted-self-application expr)
        (list expr (list 'quote expr)))
    (if (good-side? (make-quoted-self-application expr))
        <not-ok-expr> <ok-expr>)))
```

2(b) A property of Scheme expressions is *nontrivial* iff there is some expression ⟨ok-expr⟩ which has the property and some expression ⟨not-ok-expr⟩ which does not. Conclude that no nontrivial, evaluation-invariant property of Scheme expressions is decidable.

2(c) Give two "interesting" simple examples, besides the Halting Property, of nontrivial, evaluation-invariant properties of Scheme expressions. Give two "interesting" simple examples of nontrivial properties which are decidable.