CIS 425—Assignment #2
due in class, October 10

Reading

1. Mitchell: Chapters 2 (Lisp), 3.2 (λ-calculus), and 3.4 (functional programming).
2. Any basic Scheme reference.
3. McCarthy Lisp paper.

Problems

1. ................................................................. λ-Calculus
Using the definitions of substitution (see Mitchell, p. 61), verify the following equalities:
   (a) \([u/x](\lambda u.x) = (\lambda z.u)\)
   (b) \([u/x](\lambda u.u) = (\lambda z.z)\)
   (c) \([u/x](\lambda y.x) = (\lambda y.u)\)
   (d) \([u/x](\lambda y.u) = (\lambda y.u)\)

2. ................................................................. More λ-Calculus
Verify the following equalities:
   (a) \(SIII =_\beta I\), where \(S = \lambda yz.(xz)(yz)\) and \(I = (\lambda x.x)\).
   (b) \(\text{twice(twice)}fx =_\beta f(f(f(x)))\), where \(\text{twice} = \lambda fx.f(f(x))\).
   (c) Draw a diagram showing the possible reductions from \(\text{twice(twice)}fx\) to the normal form \(f(f(f(x)))\).

3. ................................................................. Fixed-points and Combinators
Let \(Z\) be the λ-term \(\lambda z.\lambda x.x(zzx)\).
   (a) Show that the term \(ZZ\) satisfies the equality \(ZZM =_\beta M(ZZM)\) for every term \(M^*\).
   (b) Show that the term \(ZZ\) satisfies the (stronger) property \(ZZM \rightarrow^* M(ZZM)\) for every term \(M\).

4. ................................................................. Order of Evaluation
In pure λ-calculus, the order of evaluation of subexpressions does not effect the value of an expression. The same is essentially true for Pure Lisp: if a Pure Lisp expression has a value under the ordinary Lisp interpreter, then changing the order of evaluation of subterms cannot produce a different value. If we declare a function \(f\) by writing something like

\[
\text{(define f (lambda (x y z) (cons (car x) (cons (car y) (cdr z)))))}
\]

then the ordinary evaluation order for a Lisp expression

\[
(f\ e_1 \ldots e_n)
\]

will be to evaluate \(e_1 \ldots e_n\) from left to right and then pass this list of values to function \(f\).

Give an example of Lisp or Scheme expressions \(e_1\ e_2\ e_3\), possibly using functions \rplaca\ or \rplacd\ with side effects, so that evaluating arguments from left to right gives a different result from evaluating them from right to left. (You may put the call to \(f\) inside some larger context, if your expressions require it.)

\*This shows that \(ZZ\) is a fixed-point combinator
For this part of the assignment, you should use Scheme (a dialect of Lisp). We have a version installed at /local/apps/chez-6.0/bin/scheme. Turn in your source code and a test run of each solution. Please note that handwritten solutions are not acceptable.

(o) Write a function \texttt{nats-list} that takes an integer argument \texttt{n} and returns a list of the natural numbers up to \texttt{n}.

\begin{verbatim}
> (nats-list 15)
(1 2 3 4 5 6 7 8 9 10 11 12 13 14 15)
\end{verbatim}

\textit{Hint: Instead of doing everything in one shot, try implementing this with a second auxiliary function.}

(b) Write a function \texttt{make-notmult-pred} that takes an integer argument \texttt{n} and returns a function of a single argument \texttt{x}, which checks that \texttt{x} is \textit{not} a multiple of \texttt{n}.

\begin{verbatim}
> (define not5 (make-notmult-pred 5))
> (not5 15)
#f
> (not5 14)
#t
\end{verbatim}

(c) Write a function \texttt{filter} of two arguments: a function \texttt{pred} (representing a predicate) and a list of numbers, \texttt{ls}. The function should return a list of all the numbers in \texttt{ls} satisfying \texttt{pred}.

\begin{verbatim}
> (filter even? (nats-list 40))
(2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40)
> (filter odd? (filter even? (nats-list 1000)))
()
\end{verbatim}

(d) Write a function \texttt{primes-list}, which takes an integer argument \texttt{n} and returns a list of all the prime numbers less than or equal to \texttt{n}.

\begin{verbatim}
> (primes-list 1)
()
> (primes-list 50)
(2 3 5 7 11 13 17 19 23 29 31 37 41 43 47)
\end{verbatim}

The prime numbers less than some fixed integer \texttt{n} can be found using the “Sieve of Eratosthenes.” The method is essentially the following: List all positive integers less than \texttt{n} (except 1), and cross off every multiple of 2. Now choose the first number not crossed out, 3, and proceed to cross off all multiples of 3. Proceed with the first number on the resulting list, 5, and so on. Note that at every step, the list resulting from the previous step begins with a prime number. You can stop when you have crossed off all multiples of primes \texttt{p} for which \(p^2 < n\).