Reading

1. Read Mitchell, Chapter 4.1–4.2

Problems

1. (5 points) Parse Tree
   Mitchell, Problem 4.1

2. (10 points) Parsing and Precedence
   Mitchell, Problem 4.2

3. (5 points) Lambda Calculus Reduction
   Mitchell, Problem 4.3

4. (10 points) Symbolic Evaluation
   The Lisp program fragment

   (define (f x) (+ x 4))
   (define (g y) (- 3 y))
   (f (g 1))

   can be written as the following lambda expression:

   \[
   \left( \frac{\lambda f. \lambda g. f (g 1) \lambda x.x + 4 \lambda y.3 - y}{\text{main}} \right)
   \]

   Reduce the expression to a normal form in two different ways, as described below.

   (a) (5 points) Reduce the expression by choosing, at each step, the reduction that eliminates a \( \lambda \) as far to the left as possible.

   (b) (5 points) Reduce the expression by choosing, at each step, the reduction that eliminates a \( \lambda \) as far to the right as possible.
5. (10 points) Lambda Reduction with Sugar

Here is a “sugared” lambda-expression using \texttt{let} declarations:

\[
\begin{align*}
\text{let } \text{compose} &= \lambda f. \lambda g. \lambda x. f(g x) \text{ in} \\
\text{let } h &= \lambda x. x + x \text{ in} \\
((\text{compose } h) h) 3
\end{align*}
\]

The “de-sugared” lambda-expression, obtained by replacing each \texttt{let } \texttt{z} = \texttt{U} in \texttt{V} by \texttt{(\lambda z. V) U} is

\[
(\lambda \text{compose.} \\
(\lambda h. ((\text{compose } h) h) 3) (\lambda x. x + x)) \\
(\lambda f. \lambda g. \lambda x. f(g x))
\]

This is written using the same variable names as the \texttt{let}-form in order to make it easier to read the expression.

Simplify the desugared lambda expression using reduction. Write one or two sentences explaining why the simplified expression is the answer you expected.

6. (20 points) Defining Terms in Lambda Calculus

In class we defined Church numerals and booleans, and showed how to define more complex functions in the pure lambda calculus. Please show how to define the following functions in the pure lambda calculus.

(a) (5 points)
Define a function \texttt{Minus} such that \texttt{Minus m n} = \texttt{m - n} if \texttt{m} > \texttt{n} and \texttt{0} otherwise. Do not use recursion, but instead define it directly.

(b) (5 points)
Define a function \texttt{LessThan} such that \texttt{LessThan m n} = \texttt{true} iff \texttt{m} < \texttt{n}.

(c) (5 points)
Define the fibonacci function \texttt{fib} such that \texttt{fib 1} = \texttt{1}, \texttt{fib 2} = \texttt{2}, and \texttt{fib n} = \texttt{fib (n-1)} + \texttt{fib (n-2)} for \texttt{n} > \texttt{2}.

(d) (5 points)
Use your definition of \texttt{fib} from above to calculate \texttt{fib 2}. Do it step by step, showing all of your work. You may assume that \texttt{fib 1} = \texttt{1} and \texttt{fib 0} = \texttt{1} and that \texttt{Plus 1 1} = \texttt{2} without showing all of the reduction steps. All other steps in the reduction should be shown.

7. (10 points) Fixed Points

We showed in class that every function \texttt{f} definable in the lambda calculus has a fixed point, i.e., there is a term \texttt{a} such that \texttt{f(a)} = \texttt{a}. In class we defined the function \texttt{Succ} as the successor function. I.e., \texttt{Succ n} = \texttt{n+1}. Needless to say, we don’t expect the successor function to have a fixed point. Compute the fixed point of \texttt{Succ}. What can you say about it? In particular, why doesn’t this contradict our expectations that the successor function on the natural numbers does not have a fixed point.