Your solutions to this assignment should be typeset using LaTeX; see the course website for instructions and resources. A pdf file containing the solutions should be submitted by 11:59pm on the due date. Submission instructions will be posted on the course website.

Read Pierce, Chapter 5.

1. **Warmup** (25 pts.)

   (a) Write the following λ-calculus terms in their fully parenthesized, curried forms. Change all bound variable names to names of the form a_0, a_1, a_2, ... where the first λ binds a_0, the second a_1, and so on.

   i. $\lambda x, y.\, z\ \lambda y, z.\, z\ y\ x$

   ii. $\lambda x.\, (\lambda y.\, y\ x)\ \lambda x.\, y\ x$

   iii. $(\lambda x.\, y\ \lambda y.\, x\ y)\ \lambda y.\, x\ y$

   (b) We defined capture-avoiding substitution into a lambda term using the following three rules:

$$\begin{array}{lcll} (\lambda x.\, e_0)\{e_1/x\} & = & \lambda x.\, e_0 & \\ (\lambda y.\, e_0)\{e_1/x\} & = & \lambda y.\, e_0\{e_1/x\} & (\text{where } y \neq x \wedge y \notin FV(e_1)) \\ (\lambda y.\, e_0)\{e_1/x\} & = & \lambda z.\, e_0\{z/y\}\{e_1/x\} & (\text{where } z \neq x \wedge z \notin FV(e_0) \wedge z \notin FV(e_1)) \end{array}$$

   In these rules, there are a number of conjuncts in the side conditions whose purpose is perhaps not immediately apparent. Show by counterexample that each of the above conjuncts of the form $x \notin FV(e)$ is independently necessary.

2. **Equivalence and normal forms** (15 pts.)

   For each of the following pairs of λ-calculus terms, show either that the two terms are observationally equivalent or that they are not. Note that for part (a), we are assuming the following definitions:

$$\begin{array}{lcl} 0 & \triangleq & \lambda s.\, \lambda z.\, z \\ 1 & \triangleq & \lambda s.\, \lambda z.\, s\ z \\ \mathsf{succ} & \triangleq & \lambda n.\, \lambda s.\, \lambda z.\, s\ (n\ s\ z) \end{array}$$

   (a) (succ 0) and 1

   (b) $\lambda x.\, x\ y$ and $\lambda x.\, y\ x$

3. **Encoding arithmetic** (20 pts.)

   Pierce (Section 5.2, *Church Numerals*) presents one way to represent natural numbers in the λ-calculus. However, there are many other ways to encode numbers. Consider the following definitions:

$$\begin{array}{lcl} \mathsf{tru} & \triangleq & \lambda x.\, \lambda y.\, x \\ \mathsf{fls} & \triangleq & \lambda x.\, \lambda y.\, y \\ 0 & \triangleq & \lambda x.\, x \\ n+1 & \triangleq & \lambda x.\, (x\ \mathsf{fls})\ n \end{array}$$

   (a) Show how to write the pred (predecessor) operation for this number representation. Reduce (pred (pred 2)) to its $\beta\eta$ normal form, which should be the representation of 0 above. pred need not do anything sensible when applied to 0.

(b) Show how to write a λ-term $\mathsf{zero?}$ that determines whether a number is zero or not. It should return tru when the number is zero, and fls otherwise. Use the definitions of tru and fls given above.

4. **Encoding lists** (15 pts.)

Pierce (Section 5.2, *Pairs*) shows how to implement pairs with a pair constructor pair, defined as $\mathsf{pair} = \lambda x.\, \lambda y.\, \lambda b.\, b\ x\ y$. Or equivalently, we could define pair by writing $\mathsf{pair}\ x\ y = \lambda b.\, b\ x\ y$. Lists can be implemented using pairs based roughly on the following idea (similar to a *tagged union*). If the list is non-empty (i.e., $\mathsf{cons}\ h\ t$, a cons cell with a head and a tail), we would like represent it as a pair of (i) a tag to remember that it is a cons cell and (ii) a pair that contains the head and tail of the list. If the list is empty (i.e., nil, the null list), we would like to represent it as a pair of (i) a tag to remember that it is nil and (ii) some arbitrary value (we don't care what).

(a) Show how to implement nil, cons, and $\mathsf{nil?}$ with the property that $\mathsf{nil?}\ \mathsf{nil} = \mathsf{tru}$ and $\mathsf{nil?}\ (\mathsf{cons}\ h\ t) = \mathsf{fls}$ for any h, t.

(b) Show how to implement the functions head and tail that when applied to a non-empty list return the head and tail of the list, respectively.

5. **S and K combinators** (25 pts.)

Consider the following definitions:

$$\begin{aligned} \mathbf{S} &\triangleq \lambda x, y, z.\, (x\ z)\ (y\ z) \\ \mathbf{K} &\triangleq \lambda x, y.\, x \end{aligned}$$

In this problem you will show that any λ-calculus expression can be expressed as a series of applications of the $\mathbf{S}$ and $\mathbf{K}$ combinators. In particular, if we think of $\mathbf{S}$ and $\mathbf{K}$ as part of the syntax, we can remove all of the λ's from the lambda calculus!

(a) Show that the $\mathbf{S}$ and $\mathbf{K}$ combinators can be used to construct an expression with the same normal form (under β and η reductions) as the identity expression $\lambda x.\, x$.

(b) Now consider the following target language, which we might call "the λ-less calculus":

$$\epsilon \ ::= \ \mathbf{S} \mid \mathbf{K} \mid x \mid \epsilon\ \epsilon$$

Write an *abstraction* function $\mathcal{A}$ such that $\mathcal{A}[\![x, \epsilon]\!]$ is extensionally equivalent to $\lambda x.\, \epsilon$ (with the definitions of $\mathbf{S}$ and $\mathbf{K}$ given above). For example,

$$\mathcal{A}[\![x, x']\!] \ = \ (\mathbf{K}\ x') \quad (\text{where } x \neq x')$$

because for all z,

$$(\mathbf{K}\ x')\ z \ = \ x' \ = \ (\lambda x.\, x')\ z$$

(c) Use $\mathcal{A}$ to construct a translation $\mathcal{C}$ from the complete λ-calculus to the λ-less calculus. Is your translation the most compact encoding possible?

Bonus factoid: We can define another combinator

$$\mathbf{X} \triangleq \lambda x.\, x\ \mathbf{K}\ \mathbf{S}\ \mathbf{K}$$

which can represent all closed λ-calculus expressions, because $\mathbf{K}$ has the same normal form as $(\mathbf{X}\ \mathbf{X})\ \mathbf{X}$ and $\mathbf{S}$ has the same normal form as $\mathbf{X}\ (\mathbf{X}\ \mathbf{X})$. So any λ-calculus term can be represented as a tree of applications of just this term!