You can turn in handwritten solutions to this part of the assignment. Please write clearly and use standard-sized (8.5 by 11in) paper. Solutions should be submitted before the beginning of class on the due date. If you choose to typeset your solutions using LaTeX, you may find the mathpartir.sty package useful.

Read Pierce, Chapter 5.

1. **Memo from PS1** Fix your memo from problem set 1 in response to the feedback.

2. **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.

3. **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:

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

   (a) $(\mathsf{succ}\ 0)$ and 1

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

4. **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:

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

(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 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.

5. **Encoding lists** (25 pts.)

Pierce (Section 5.2, *Pairs*) shows how to implement pairs with a pair constructor pair, defined as pair $= \lambda x.\, \lambda y.\, \lambda b.\, b\ x\ y$. Or equivalently, we could define pair by writing 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., 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 nil? with the property that nil? nil = tru and nil? (cons $h\ t$) = 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.

6. **Translation and conjectures in Redex** (15 pts.)

(a) Formalize the λ-calculus and a call-by-value operational semantics. Call this language Lam.

$$e \quad ::= \quad x \ \mid\ \lambda x.\, e \ \mid\ e_1\ e_2$$

(b) Formalize the λ-calculus with booleans and a call-by-value reduction relation. Call this language LamBool.

$$e \quad ::= \quad x \ \mid\ \lambda x.\, e \ \mid\ e_1\ e_2 \ \mid\ \mathsf{true} \ \mid\ \mathsf{false} \ \mid\ \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$$

(c) Define a metafunction `translate` that translates LamBool to Lam. Decide what language to define this metafunction on and explain your choice in your README.

Hint: You can define a union language ST (source+target) for this purpose. Here are two ways to define this union language:

(define-union-language ST LamBool Lam)

(define-union-language ST (s. LamBool) (t. Lam))

Look up define-union-language in the Redex reference manual to understand the difference and decide which one to use.

(d) Formalize a conjecture that the above translation is correct and test your conjecture using `redex-check`. Informally, the "correctness" property we are interested in says that (1) if a source expression e_S diverges, then its translation diverges and (2) if a source expression e_S evaluates to a value v_S, then the translation of e_S should evaluate to some v_T that is "equivalent" to v_S.